MODELLING, TRANSFORMATIONS, AND SCALING DECISIONS IN CONSTRAINED OPTIMIZATION PROBLEMS

John Joseph Timar

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MODELLING, TRANSFORMATIONS, AND SCALING DECISIONS IN CONSTRAINED OPTIMIZATION PROBLEMS

by

John Joseph Timar

March 1976

Thesis Advisor: G. H. Bradley

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Modelling, Transformations, and Scaling Decisions in Constrained Optimization Problems		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis March 1976 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) John Joseph Timar		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE March 1976 13. NUMBER OF PAGES 97
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
separable programming, goal programming, linear fractional programming, mathematical programming, nonlinear programming, nonlinear optimization, transformations, scaling, rotation of coordinates, translation of coordinates, quadratic diagonalization, SUMT, GRG.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis investigates various modelling choices and modelling decisions that can be used by defense analysts when solving nonlinear optimization problems. A discussion is given of separable programming, goal programming, and linear fractional

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |
(Page 1)

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract (Cont'd.)

programming models, and a description of the manner by which they can be converted to equivalent linear programs. Transformations of variables recommended in the literature are tested on several well-known test problems using GRG and SUMT nonlinear programming codes. The sensitivity of the GRG code to scaling, rotation of coordinates, and translation of variables is examined. Transformations to obtain separability of variables and experiments using a diagonalization algorithm to transform quadratic expressions into sums of squares are discussed. Barrier and penalty function transformations are also considered.

Modelling, Transformations, and Scaling Decisions in Constrained Optimization Problems

by

John Joseph Timar
Lieutenant, United States Navy
B.S., Lehigh University, 1968

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
March 1976

ABSTRACT

This thesis investigates various modelling choices and modelling decisions that can be used by defense analysts when solving nonlinear optimization problems. A discussion is given of separable programming, goal programming, and linear fractional programming models, and a description of the manner by which they can be converted to equivalent linear programs. Transformations of variables recommended in the literature are tested on several well-known test problems using GRG and SUMT nonlinear programming codes. The sensitivity of the GRG code to scaling, rotation of coordinates, and translation of variables is examined. Transformations to obtain separability of variables and experiments using a diagonalization algorithm to transform quadratic expressions into sums of squares are discussed. Barrier and penalty function transformations are also considered.

TABLE OF CONTENTS

I. INTRODUCTION

In the defense community, military and civilian analysts are frequently confronted with problems in which one or more objectives are to be optimized subject to resource constraints. By their very nature, many military problems are very complex and highly unstructured. As a result, the analyst is faced with many decisions including: what is a suitable measure of effectiveness; what are the effective constraints; what assumptions need to be made; can the original problem be reduced or transformed to a simpler model which is easier to solve; how sensitive are the results to the underlying assumptions?

It is the concern of this thesis to investigate various modelling choices and various modelling decisions in order to guide the analyst. Modelling choices include the actual form of the underlying mathematical model, for example, goal programming, separable programming, or linear fractional programming. Modelling decisions describe the operations that are performed on the mathematical model once a specific formulation is chosen. Included in this category are transformation of variables, scale changes, translation and rotation of coordinates. Chapter II will introduce terminology and classification of mathematical programming

problems; Chapter III will illustrate some modelling choices; Chapters IV and V will describe the various transformations and scaling operations examined in this thesis.

The scope of the research involved in this study is an examination of the applicable mathematical programming literature regarding modelling, transformations, and scaling, followed by the testing of specific ideas on two commercially available nonlinear programming codes. The literature search revealed that little groundwork has been done in this area, most of what has been found is dated; and, furthermore, no work of this kind has been done on commercial constrained optimization codes. The two codes in question are Generalized Reduced Gradient (GRG) of Lasdon, Waren, Ratner, and Jain [1] [2], and Sequential Unconstrained Minimization Technique (SUMT) of Fiacco and McCormick [3] [4] [5]. System documentation for this GRG code was dated November, 1975 [6].

For an analyst who has not gained much experience with constrained optimization, the chapter on specific modelling choices will illustrate the types of conversions that can be made on nonlinear problems to get them into a form where a commercial linear programming package can be used.

The main thrust of the computational experiments was to take a few well known test problems for which the optimum solution was known, transform or scale the problem in some

manner, and determine what effect the change had on the code. It needs to be emphasized that SUMT and GRG are just two of several constrained optimization codes that have been developed in recent years. Without codes of this type, the analyst can still solve mathematical programs containing nonlinearties by using a linear program that approximates the nonlinear terms. The state of the art in unconstrained nonlinear optimization techniques is highly developed, but in constrained nonlinear optimization, although the body of theory is large, the area of technique has evolved slowly.

The transformations of variables discussed in Chapter IV can be used to transform some nonlinear problems into unconstrained or partially constrained problems. If only a few constraints can be eliminated from a problem, it can make problems easier to solve using the SUMT code. On the other hand, empirical results presented in this thesis indicate that the same transformations, when applied to test problems solved by GRG or SUMT, made the test problems more difficult to solve. Computation time was considerably increased using transformations, and in numerous experiments, the GRG code could not find a feasible point. Since these transformations actually restrict the variables to be nonnegative or take on values in a certain range, a given problem can be modified significantly by a change of

variables. In addition, transformations may cause some local optima to be lost.

Transformations to obtain separability of variables is also discussed in Chapter IV and includes a description of a diagonalization algorithm to transform quadratic expressions into sums of squares. The results of experiments that were conducted in diagonalizing quadratic forms of different dimensions are included to give the analyst an idea of the time trade-off that he must make to get such an expression into separable form. There is also a discussion of barrier and penalty function transformations, of which the SUMT code used in the experiments is a prime example.

The final chapter examines the use of scaling, rotation, and translation operations and discusses the sensitivity of the GRG code to these techniques. The empirical results from the test problems considered indicate that proper scaling is critical to the successful utilization of GRG. This code is also highly affected by attempts at rotation and translation of coordinates. Multiple rotation operations on a test problem having nonlinear equality constraints made that problem much harder to solve. The translation experiments indicate that translations will increase the amount of time and the number of iterations required to find a solution.

The principles, maxims, and heuristics presented here will not guarantee success, but if adhered to, should provide the analyst with guidance when attempting to solve constrained optimization problems. The codes do not guarantee the correct optimum solution, and in some cases, will not construct a solution even though one may exist. Each computer code has a number of adjustable prameters, and these adjustments in turn, affect the efficiency of the corresponding algorithm. There are recommended average values for the parameters required by a specific code, but choosing these values may prevent the code from operating at maximum efficiency for a given problem. A final caution on the codes is in order. They have been fine tuned on a number of well-known properly scaled test problems of varying degrees of difficulty. The performance of these codes on a real-world problem is highly dependent upon correct formulation and proper scaling by the analyst.

As a prelude to the remainder of this study, consider the weapon allocation problem developed by Koopman /7/, which is an example of a nonlinear optimization problem. The basic model is one that maximizes the expected damage subject to the total number of weapons available, and in its simplest mathematical form can be expressed as:

$$\text{maximize} \quad \sum_{j=1}^{n} V_j \left[1 - \prod_{j=1}^{m} \exp\left(-\mu_{ij} x_{ij} \right) \right] \qquad (1)$$

$$\text{subject to:} \quad \sum_{j=1}^{n} x_{ij} \leq N_i \qquad i=1,2,.....m$$

$$x_{ij} \geq 0$$

where:

V_j = value of j^{th} target

x_{ij} = number of weapons of type i allocated to target j

N_i = total number of weapons of type i

μ_{ij} = constant which incorporates the probability of hitting target j with weapon i, and the rate at which target value decreases with each direct hit

This allocation problem will be referred to again in later chapters. An analyst trying to solve this problem can either apply transformations to get the problem into separable form, or try to solve it directly with a code like GRG or SUMT. His approach will be greatly influenced by the programming codes available to him.

II. THE MATHEMATICAL PROGRAMMING PROBLEM

A. DEFINITION OF THE PROBLEM

Let $\underline{x} = (x_1, x_2, \ldots\ldots, x_n)^T$ represent a vector in a space of n dimensions. Let $f(\underline{x})$, and $g_j(\underline{x})$ $j=1,2,\ldots m$, be functions defined in the vector space. The general mathematical programming problem is to find a $\underline{x}^*$ such that $f(\underline{x}^*)$ will be the maximum or minimum of $f(\underline{x})$ under the constraining conditions:

$$g_j(\underline{x}^*) \leq 0 \qquad j=1,2,\ldots\ldots,m$$

B. TERMINOLOGY

Before proceeding, it is necessary to specify some terminology appropriate to the mathematical programming problem that is used throughout this thesis.

The $\underline{x}$ vector is an n-dimensional vector of unknown problem variables. Specification of an $\underline{x}$ vector determines a point in n-dimensional space and also determines a value for $f(\underline{x})$ and $g_j(\underline{x})$.

The $g_j(\underline{x})$ in inequality (2) are called constraints, and form a closed region in n-dimensional space, thus limiting values of $\underline{x}$ to points in the closed region.

The closed region defined by the constraints is called the feasible region for a given problem. A feasible point is any point $\underline{x}$ that lies in the feasible region. Points outside this region are called non-feasible.

The function $f(\underline{x})$ is called the objective function of the problem. Mathematical programming attempts to optimize $f(\underline{x})$ over the feasible region defined by inequality (2).

Optimal solutions are not necessarily unique. More than one solution may have the same minimum or maximum value. Necessary conditions for optimality are conditions that an optimal solution must satisfy, but that other nonoptimal solutions may also satisfy. A sufficient condition for optimality is one that, if satisfied by a given solution, guarantees that the given solution is optimum. For many general problems, conditions that are both necessary and sufficient cannot be determined; the best that can be done is to show that a given solution is a local optimum. A local minimum (maximum) is any feasible point such that any small perturbation around it, still remaining in the feasible region, will increase (decrease) the value of the objective function. The global optimum is that local optimum for which the objective function has its smallest (largest) value.

C. CLASSIFICATION OF MATHEMATICAL PROGRAMMING PROBLEMS

Mathematical programming problems are generally classified on the basis of the mathematical form of $f(\underline{x})$ and $g_j(\underline{x})$.

Linear programming results when both $f(\underline{x})$ and the $g_j(\underline{x})$ are linear functions of $\underline{x}$. A linear function has constant values for its partial derivatives with respect to $\underline{x}$, and thus has a constant gradient. Linear programming is the most well known mathematical programming problem, and computer routines using the simplex algorithm are widely available and can handle thousands of variables and constraints.

Goal programming is a simple extension of linear programming which attempts to handle multiple, and frequently conflicting goals. The goal programming formulation is discussed in the next chapter. The goal programming approach is to combine the multiple goals, weighted by appropriate factors, into a single objective function that is to be optimized. Lee /8/ /9/ has made numerous applications of goal programming dealing specifically with the problem of handling a hierarchy of goals, in which the most important goal has a much higher priority than lower level goals.

When $f(\underline{x})$ and/or the $g_j(\underline{x})$ are not linear functions of $\underline{x}$, the problem is called a nonlinear programming problem. Definitive general statements of necessary and sufficient conditions are available only for limited cases. For the special case of a convex nonlinear programming problem in which the objective function and constraints are convex, the necessary conditions that an optimal solution must

satisfy are commonly referred to as the Kuhn-Tucker conditions. References [10] [11] [12] [13] provide good descriptions of the Kuhn-Tucker conditions.

When $f(\underline{x})$ is a quadratic function of $\underline{x}$ and the $g_j(\underline{x})$ are linear, the problem is called a quadratic program. In principle, this problem is almost as easy to solve as the linear programming problem, and differs from it mainly in that the gradient of $f(\underline{x})$ is a linear function of the $\underline{x}$. In practice, the quadratic programming problem is solved either by conversion to an approximate linear programming problem, or by solving it directly using a nonlinear programming algorithm.

If the constraints are linear and the nonlinear objective function can be expressed as the ratio of two linear functions, a special mathematical programming problem known as linear fractional programming results. It can be solved by computing the optimum solution to at most two linear programs.

Many nonlinear problems can be solved by the use of some linearizing technique followed by the application of the simplex algorithm. One such technique is called separable programming which requires that a nonlinear function be separated into a sum of several terms, each of which is a function of a single variable. These terms are linearized

by calculating their values over a grid of points in the convex region. The simplex algorithm is then applied to the linearized problem.

Separable programming, goal programming, and linear fractional programming can be converted into problems solvable by the simplex algorithm, and will be discussed, in turn, in the next chapter.

III. MODELLING CHOICES

The previous chapter was intended to present an overview of the modelling choices available to the analyst to model a given mathematical programming problem. As the analyst formulates his model he must bear in mind the tradeoff between simplicity and accuracy. If he oversimplifies his model by assuming away nonlinearities, he may wind up with a linear programming model that gives poor results and is not representative of the actual problem. If he attempts to include every detail of the problem, the formulation may become so complex that the model becomes incomprehensible.

The weapon allocation model cited in Chapter I is an example of a simple model formulation that can easily become a formidable problem by making a few modifications. A descriptive problem studied by Bracken and McGill [14] illustrates how difficult this problem can become. The application involves the targeting of sea-launched ballistic missiles on strategic bomber bases. The objective is to allocate submarines to possible launch areas and to find a targeting pattern against the bomber bases so as to maximize the numbers of bombers destroyed. There are technological constraints which prevent launching all of the missiles simultaneously. Furthermore, the flight time of

a missile depends upon the distance between the launch point and the target. Consequently the enemy can scramble some of his bombers, with the number scrambled increasing with time. The objective function will have to be modified further if the missiles destroy only part of the bomber bases. These features of time-phased allocations and time deteriorating values significantly alter the model formulation. The same authors offer other interesting defense applications in reference [15].

There is a direct interaction between the mathematical programming problem and the techniques available to solve it. The state of the art in nonlinear programming is such that very large problems can be handled only in special cases for problems having special structure. The generally available codes are currently limited to about 100 variables because of excessive computational time and excessive storage requirements. There is also a tradeoff between obtaining an exact solution to an approximate problem, or an approximate solution to an exact problem.

In this chapter three special purpose nonlinear programming problems are discussed that can be solved by linear programming methodology. Separable programming is the only one of the three that is widely known. Since it uses large scale linear programming codes, it can be used to solve

nonlinear problems having thousands of variables and constraints. Goal programming and linear fractional programming are two other models that can also be solved by commercial linear programming codes. However, the selection of one of these codes is not necessarily the best approach to use for a given problem if the problem is of moderate size. These codes do have the following advantages:

1. They are easy to prepare.
2. They can be of very large size.
3. They can be used routinely in that they are solvable by generally available, well-documented linear programming codes.
4. It is easy to perform a sensitivity analysis on the variables and/or parameters of the problem to determine the effect of changes in these quantities.

The discussion of these models in this chapter should provide a framework for the analyst that will assist him in choosing a specific model.

A. SEPARABLE PROGRAMMING

Separable programming is used to obtain an approximate solution to nonlinear problems having a separable objective function and constraints. A separable function is one that can be written in the form $f(\underline{x}) = \sum_i f_i(x_i)$, where $f_i(x_i)$ is a function of a single variable x_i. The mathematical formulation is:

$$\text{maximize} \quad \sum_{i=1}^{n} f_i(x_i) \tag{3}$$

$$\text{subject to:} \quad \sum_{i=1}^{n} g_{ij}(x_i) \leq b_j \qquad j=1,2,\ldots,m$$

$$x_i \geq 0$$

The separable problem is reduced to a linear problem by approximating each separable function by a piecewise linear function. There are several excellent references that provide a thorough description of the procedure for converting the separable problem to an approximate linear problem. Dantzig [11], Hadley [13], Miller [16], and Beale [17] [18] are included on this list.

The weapon system allocation problem (1) can be formulated as a separable programming problem by introducing the new variable $z_j = \sum_{i=1}^{m} \mu_{ij} x_{ij}$. The problem therefore becomes:

$$\text{maximize} \quad \sum_{j=1}^{n} v_j(1-e^{-z_j}) \tag{4}$$

$$\text{subject to:} \quad z_j - \sum_{i=1}^{m} \mu_{ij} x_{ij} = 0 \qquad j=1,2,\ldots,n$$

$$\sum_{j=1}^{n} x_{ij} \leq N_i \qquad i=1,2,\ldots,m$$

$$x_{ij} \geq 0$$

or equivalently,

$$\text{minimize} \quad \sum_{j=1}^{n} V_j e^{-z_j} \tag{5}$$

$$\text{subject to:} \quad z_j - \sum_{i=1}^{m} \mu_{ij} x_{ij} = 0 \qquad j=1,2,....n$$

$$\sum_{j=1}^{n} x_{ij} \leq N_i \qquad i=1,2,....m$$

$$x_{ij} \geq 0$$

Problem (5) can now be approximated by piecewise linear functions. A section in the following chapter on transformations describes various methods of converting nonlinear terms of several variables into separate terms of single variables.

Some motivations for dealing with separable programming are as follows:

(1) in the case of convex problems, it allows the use of large scale linear programming codes;

(2) in the case of nonconvex problems, it allows the use of linear programming codes with a separable option, for which there are special basis entry rules (see Beale /17/ for a discussion of these rules);

(3) it is easy to adapt the separable formulation to a constrained nonlinear programming code such as SUMT (i.e., it is much simpler to compute the analytical derivatives

required by SUMT when the variables are separated);

(4) by removing the interaction between variables, it is easy to see the effects of transformations on the variables.

B. GOAL PROGRAMMING

Goal programming is a useful concept when multiple goals are either in direct conflict, or can be achieved only at the expense of other goals. A simple example would be a model that involves two types of manpower in two different time periods. Assume that an analyst is faced with the task of recommending the number of officers and enlisted personnel to recruit for a special program in the next two fiscal years. Assume that the only costs involved are the salaries of the recruits, which are $15,000 for officers and $10,000 for enlisted men. The budget for the program is $4,000,000 for the first fiscal year; $6,000,000 for the second. The desired goals for the number of officers is 50 for the first year, 75 for the second year. The corresponding minimums are 10 and 60 for the two years. The desired goals for enlisted men in the program are 250 and 400, with corresponding minimums of 100 and 175.

A goal programming formulation is:

$$\text{minimize } |x_1-50| + |x_2-250| + |x_3-75| + |x_4-400|$$

subject to: $x_1 \geq 10$

$x_2 \geq 100$

$x_3 \geq 60$

$x_4 \geq 175$ (6)

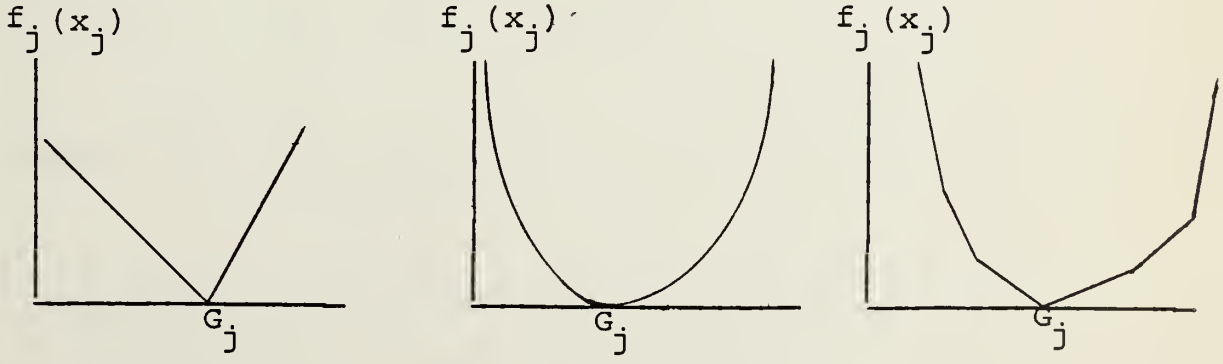
$15x_1 + 10x_2 \leq 4000$

$15x_3 + 10x_4 \leq 6000$

where
- x_1 = number of officers in the first year
- x_2 = number of enlisted men in the first year
- x_3 = number of officers in the second year
- x_4 = number of enlisted men in the second year

Figure 1 illustrates several types of objective functions which can be classified as goal programming problems. Figure 1-A is an absolute value function having asymmetric weights. Figure 1-B is a quadratic function $(x_j - G_j)^2$, and Figure 1-C is a piecewise linear function.

FIGURE 1: Typical Goal Programming Models



(A) (B) (C)

The goal objective function consists of a sum of functions of a single variable, some of which may be nonlinear. Therefore, the goal programming problem is reducible to either a linear program or separable program depending on how the model is formulated.

A more general formulation of the goal programming problem is:

$$\begin{aligned} &\text{minimize} && f(\underline{x}) = \sum_j f_j(x_j) \\ &\text{subject to:} && \underline{A}\,\underline{x} \leq \underline{b} \\ & && \underline{x} \geq 0 \end{aligned} \tag{7}$$

If the absolute value function is used, the mathematical formulation is:

$$\begin{aligned} &\text{minimize} && \sum_{j=1}^{n} w_j \left| \underline{c}^T \underline{x} - G_j \right| \\ &\text{subject to:} && \underline{A}\,\underline{x} \leq \underline{b} \\ & && \underline{x} \geq 0 \end{aligned} \tag{8}$$

If a quadratic function is used, the formulation is:

$$\begin{aligned} &\text{minimize} && \sum_{j=1}^{n} w_j \left(\underline{c}^T \underline{x} - G_j \right)^2 \\ &\text{subject to:} && A\,\underline{x} \leq b \\ & && \underline{x} \geq 0 \end{aligned} \tag{9}$$

A more extensive mathematical treatment of goal programming can be found in Refs. [19] and [20].

To handle the nonlinear problem described in (8), slack variables y_j and z_j are added to the problem to represent

positive and negative deviation from each goal G_j. For each goal, one or both of these slack variables will equal zero. Minimizing the sum of absolute deviations is equivalent to the following linear program:

$$\text{minimize} \quad \sum_{j=1}^{n} w_j \left| y_j + z_j \right| \tag{10}$$

$$\text{subject to:} \quad \underline{c}^T \underline{x} - y_j + z_j = G_j \qquad j=1,2,....n$$

$$\underline{A}\,\underline{x} \preceq \underline{b}$$

$$\underline{x}, \underline{y}, z \geq 0$$

The nonlinear problem described in (9) can be solved by making piecewise linear approximations to each such term in the objective function, and using the technique of separable programming.

The manpower planning model described previously can be reformulated as the following linear program:

$$\text{minimize} \quad y_1+z_1+y_2+z_2+y_3+z_3+y_4+z_4$$

subject to: (11)

$$x_1 - y_1 + z_1 = 50 \qquad x_1 \geq 10$$

$$x_2 - y_2 + z_2 = 250 \qquad x_2 \geq 100$$

$$x_3 - y_3 + z_3 = 75 \qquad x_3 \geq 60$$

$$x_4 - y_4 + z_4 = 400 \qquad x_4 \geq 175$$

$$15 x_1 + 10 x_2 \leq 4000 \qquad x_i, y_i, z_i \geq 0$$

$$15 x_3 + 10 x_4 \leq 6000 \qquad i=1,....,4$$

A real world example of a goal programming model is one that was done by Major Calvin Anderson of the U.S. Army Concepts Analysis Agency, and by professors G.H. Bradley and G.G. Brown of the Naval Postgraduate School. The model determined the optimum distribution of officers, by rank, in various specialties, with the ideal utilization equal to 0.5 in primary and secondary specialties. The total number of variables was 437, and the objective was to minimize the sum of deviations between the actual and desired utilization in the specified billets. The problem was solved using both absolute value and piecewise linear (6 to 100 segments per function approximated) approximations to a quadratic goal function on a FORTRAN network code called GNET [21]. This asset utilization model illustrates both goal and separable programming concepts. In increasing the number of segments to approximate each function from 2 to 6-100, the number of arcs in the model increased from 874 to 4334, and the solution time increased from 2.75 to 36.87 seconds.

C. LINEAR FRACTIONAL PROGRAMMING

Fractional programs result when rates such as target value destroyed to weapons expended, or retention rate of aviators over a time planning horizon are to be optimized. The general mathematical formulation is:

$$\text{maximize} \quad \frac{f(\underline{x})}{q(\underline{x})}$$

$$\text{subject to:} \quad g_j(\underline{x}) \leq 0 \qquad j=1,2,.....m \tag{12}$$

An excellent theoretical development of fractional programming can be found in Ref. [22]

If f and q are linear and the constraints are linear, then (12) is called a linear fractional program, and can be written as:

$$\text{maximize} \quad \frac{\underline{c}^T\underline{x} + \alpha}{\underline{d}^T\underline{x} + \beta}$$

$$\text{subject to:} \quad \underline{A}\,\underline{x} \leq \underline{b}, \quad \underline{x} \geq 0 \tag{13}$$

Where α and β are real numbers.

The linear fractional program (13) can be reduced to a linear program by the following variable transformation proposed by Charnes and Cooper [23]:

$$\underline{y} = \frac{1}{\underline{d}^T\underline{x} + \beta}\,\underline{x} \tag{14}$$

$$\underline{z} \quad \frac{1}{\underline{d}^T\underline{x} + \beta}, \qquad \text{if } \underline{d}^T\underline{x} + \beta > 0$$

The resulting linear program is:

$$\text{maximize} \quad \underline{c}^T\underline{y} + \alpha\,\underline{z}$$

$$\text{subject to:} \quad \underline{A}\,\underline{y} - \underline{b}\,\underline{z} \leq 0, \quad \underline{d}^T\underline{y} + \beta\,\underline{z} = 1, \quad \underline{y} \geq 0,\ \underline{z} > 0 \tag{15}$$

If $\underline{d}^T \underline{x} + \beta < 0$, this transformation will not work but can be modified easily as follows:

$$\underline{y} = - \frac{1}{\underline{d}^T \underline{x} + \beta} \underline{x}$$

$$\underline{z} = - \frac{1}{\underline{d}^T \underline{x} + \beta} \quad , \quad \text{if } \underline{d}^T \underline{x} + \beta < 0 \tag{16}$$

The new linear program to solve becomes:

$$\text{maximize} \quad \underline{c}^T \underline{y} + \alpha \underline{z}$$

$$\text{subject to:} \quad \underline{A}\,\underline{y} - \underline{b}\,\underline{z} \leq 0$$

$$\underline{d}^T \underline{y} + \beta \underline{z} = - 1$$

$$\underline{y} \geq 0 \quad , \underline{z} > 0$$

The case where the denominator is allowed to be zero in the feasible region is considered in Ref. [24].

IV. PROBLEM AND VARIABLE TRANSFORMATIONS

Once the analyst decides on a mathematical model, which may be dictated by the codes available to him, he is then faced with the decision of whether to apply transformations. There are several reasons for considering the use of transformations. First, a computer program for the solution of the transformed problem may be readily available. Second, the transformed problem may require less computational time. Third, the problem as initially formulated may be too difficult to solve and thus require an approximation. Finally, the reformulated version may also provide the analyst with more insight and information than was available from the original problem.

The classification of transformations that are of interest to the analyst are the following:

(1) transformations to an unconstrained or partially constrained problem by a change of variables;

(2) transformations to a separable programming format;

(3) transformations to an unconstrained problem by barrier or penalty functions.

These transformations will be discussed in order in the following sections of this chapter along with a description of the experiments used to test them.

A. TRANSFORMATIONS TO UNCONSTRAINED OR PARTIALLY CONSTRAINED PROBLEMS BY A CHANGE OF VARIABLES

Variable transformations to eliminate constraints have received little attention in the literature, and it is one area of mathematical programming that is much in need of updating. Very little empirical work has been done in this area. In this section transformations recommended for conversion of constrained optimization problems to unconstrained or partially constrained problems are discussed. No previous work has been done with these transformations in conjunction with a generally available nonlinear programming code. These transformations are considered here because an appropriate choice can possibly eliminate some constraints and make the problem easier to solve. In the next section, the results of testing these transformations on the GRG and SUMT codes are discussed.

Constrained optimization problems can sometimes be reduced to a simpler form in which no constraints appear explicitly. These problems can then be solved by a wide variety of unconstrained optimization techniques which handle general nonlinear functions.

Box /25/ was one of the first to investigate the possibility of using transformations to eliminate linear constraints. The following table lists linear constraints and some of the change of variables transformations that can be made.

TABLE 1: Change of Variables Transformations for Linear Constraints

Constraints	Transformations
$x_i \geq 0$	$x_i = y_i^2$; $x_i = e^{y_i}$; $\lvert x_i = y_i \rvert$
$0 \leq x_i \leq 1$	$x_i = \sin^2 y_i$; $x_i = \frac{e^{y_i}}{e^{y_i}+e^{-y_i}}$
$\ell_i \leq x_i \leq u_i$	$x_i = \ell_i + (u_i - \ell_i) \sin^2 y_i$
$-1 \leq x_i \leq 1$	$x_i = \sin y_i$
$0 \leq x_i \leq x_j \leq x_k$	$x_i = y_i^2$, $x_j = y_i^2 + y_j^2$, $x_k = y_i^2 + y_j^2 + y_k^2$

If each variable in a problem has constant upper and lower bounds, $\ell_i \leq x_i \leq u_i$, then the feasible region consists of a rectilinear n-dimensional box. Replacing each x_i by $\ell_i + (u_i - \ell_i) \sin^2 y_i$ means that an unconstrained optimum in $\underline{y}$- space is being sought. The periodicity of optimal solutions in transformed space will not cause any difficulty if small step-size adjustments are made by the optimization technique.

These transformations map points in the neighborhood of $\underline{y}_0$ in $\underline{y}$- space into the neighborhood of $\underline{x}_0$ in $\underline{x}$ - space. Although they are not necessarily a 1:1 mapping, these transformations cannot introduce any additional local optima.

Experience in the application of these techniques is limited, but the possibility of their use should be kept in mind by the analyst whenever he formulates a problem. If even a few constraints can be eliminated, it should help a code such as SUMT in which the constraints are included in the modified objective function.

B. COMPUTATIONAL EXPERIENCE WITH VARIABLE TRANSFORMATIONS

There were three test problems used in the experiments on the GRG and SUMT codes. The experiments were conducted on the NPS IBM 360/67 computer system with the Fortran H Compiler. The codes were loaded on the same data cell and both used double precision arithmetic. Problems were taken from the appendix of Himmelblau's text on applied nonlinear programming /26/, and are given in Appendix A of this thesis along with the original source. They will be referred to as Himmelblau problems 16,4, and 20 respectively. Before describing the experiments performed, a few words are in order regarding the test problems. Problem 16 includes a quadratic objective function, of nine variables, 13 quadratic constraints, and one upper bound. Problem 20 includes a linear objective function of 24 variables, which are subject to 12 nonlinear equality constraints, two linear equality constraints, and six nonlinear inequality constraints. The 14 equality constraints make it a very

difficult problem. Problem 4 has a nonlinear objective function of 10 variables which is a logarithmic function. It is subject to three linear equality constraints and all variables have lower bounds. The starting point for each test problem was infeasible. The experiments were conducted so that the same starting point was always used for a given test problem. Table 2 summarizes pertinent information on the three test problems.

The GRG code used in these experiments was designed to handle small or moderate size problems containing up to 100 variables and 100 constraints, of which 60, at most, can be binding at any one time. This limitation is based upon storage requirements. Lasdon is currently working on a GRG code that will handle much larger problems. The SUMT code is limited to problems having less than 100 variables and less than 200 constraints. The GRG method is described in Appendix C, and the reader is strongly urged to review this appendix before reading the following discussion of empirical results. The SUMT method is discussed in section D of this chapter in conjunction with penalty and barrier functions.

TABLE 2: Test Problems Used with GRG and SUMT Codes

Problem Number	16	4	20
Number of variables	9	10	24
Number of equality constraints:			
linear	0	3	2
nonlinear	0	0	12
Number of inequality constraints			
linear	0	0	0
nonlinear	13	0	6
Number of lower bounds	1	10	24
Number of upper bounds	0	0	0
CPU time, seconds:			
GRG	2.90	1.68	13.56
SUMT	11.69	----	240.0+*

*Optimum still not reached when program was terminated

The GRG method is one that follows an inequality constraint very closely and therefore is very likely to terminate at a local rather than global optimum. The experiments performed using GRG and SUMT are listed in Appendix B. Experiments #2, #25, and #16 are the reference runs for problems 16, 4, and 20 respectively using the GRG code. Experiments 28 and 31 are the reference runs for problems 16 and 20 respectively using SUMT. No reference run was made for problem 4 using SUMT. The SUMT code was not used extensively in these experiments because the user must supply

the gradient and Hessian matrix in order for the code to run efficiently. It was felt that this was too time consuming (high preparation time) and error producing (debugging time excessive) to be worthwhile. GRG, on the other hand, is very easily modified to account for a change of variables. This is so because GRG uses finite differencing to evaluate gradients, although the user can provide a subroutine with the exact analytical derivatives.

In general, problems with nonlinear equality constraints and either nonlinear or linear inequality constraints are the hardest to solve, followed by linear equality constraints with nonlinear inequality constraints, and lastly, nonlinear or linear inequality constraints. Therefore, of the three test problems, the order of decreasing difficulty is 20, 4, 16.

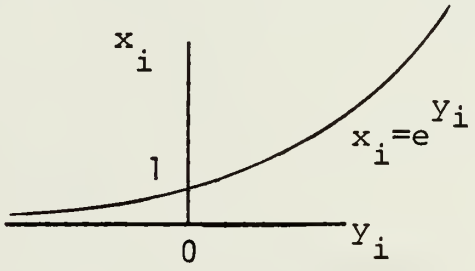
Before discussing the transformations on problem 16, it must be noted that a fixed parameter in the GRG code had to be modified before the correct optimum could be obtained. A parameter called TOL in the DEGEN subroutine had to be tightened in order to obtain the correct solution for problem 16. The subroutine DEGEN is called when the basis constructed is degenerate. The change was suggested by Lasdon for this particular test problem, and will not work for other problems. Experiments 1 and 2 show the effect of this modification on

the basic problem. Experiments 3 through 15 show the effects of various transformations on problem 16. In experiment 3, the problem was modified by placing lower bounds on all variables, but it yielded only a local optimum.

In experiment 4, the transformation $x_i^2 - y_i^2$ was attempted which again stopped at a local optimum. The problems run in experiments 3 and 4 were then run on the SUMT code (experiments 29 and 30), both yielding the correct maximum value but at different values of $\underline{x}$. Experiments 5 and 6 were then repeats of experiments 3 and 4, except for the starting points, which were taken from the optimum $\underline{x}$ in the SUMT experiments. Aside from these two experiments all others were commenced from the same starting point by modifying the input vector as appropriate.

In experiments 7, 8, and 14, the transformation $x_i = |y_i|$ was attempted. The first two of these were with and without lower bounds on y_i respectively, both yielding local optima which differed only in that x_3 and x_4 were reversed. In experiment 14, the initial $\underline{y}$ vector was the optimum vector from experiment 2(the reference run), and no lower bounds were placed on the y_i. No feasible point was found in this run because the final constraint was the only one not satisfied initially, and in attempting to satisfy it, the step size was reduced to zero causing termination.

In experiments 9 and 10, the transformation $x_i=e^{y_i}$ was tried, first without lower bounds and then with them. Running the problem without lower bounds produced an apparent local optimum, while running it with lower bounds kept the search outside the feasible region. The reason this happened is evident from a sketch of e^{y_i} which is always positive (note that $e^0=1$). Constraining $y_i \geq 0$ will always violate



constraints 1, 3, and 12.

Experiments 11, 12 and 13 involved the transformation $x_i= \sin^2 y_i$. The first two were run without and with lower bounds on y_i respectively, but both start from the same initial point which for this transformation is infeasible. To see why this is so, consider constraint 1 with x_3 replaced by $\sin^2 y_3$, and x_4 replaced by $\sin^2 y_4$. This makes the constraint:

$$1- \sin^4 y_3- \sin^4 y_4 \geq 0$$

Taking the gradient of this constraint with respect to $\underline{y}$ yields the following nonvanishing components.

$$\frac{\partial g_1}{\partial y_3} = -4(\sin^3 y_3) \quad (\cos y_3)$$

$$\frac{\partial g_1}{\partial y_4} = -4(\sin^4 y_4) \quad (\cos y_4)$$

Evaluated at $\pi/2$ radians, both terms vanish since cos $(\pi/2)=0$. Thus all elements of the gradient of constraint 1 vanish. The same holds true for all the other constraints and the objective function. In experiment 13 a different starting point was used ($y_i = \pi/4$, i=1,....9), and was successful in reaching a local optimum.

In experiment 15, the transformation $x_i = \frac{e^{y_i}}{e^{y_i} + e^{-y_i}}$ was tried with no lower bounds on y_i. This transformation constrains x_i to the interval (0,1). As with the previous transformation, this one never gets going because the reduced gradient at the starting point is zero for all 9 variables.

This same group of transformations was then tried on problem 20. As previously noted, experiment 16 is the reference run for this problem using GRG, and the optimum value of $\underline{x}$ is listed there. Experiment 17 handles the lower bounds explicitlyas inequality constraints. Although it arrives at the optimum point, it takes 70 per cent longer to run than the reference experiment.

In experiment 18, the transformation $x_i=y_i^2$ was tried, and the transformed problem succeeded in reaching the optimum point but required more than double the time of the reference run. The transformation attempted was $x_i= |y_i|$ in experiment 19. Here too, the optimum point was reached but in nearly triple the reference time. In experiment 20, the transformation $x_i= |y_i|$ was tried but with lower bounds on the variables removed. No feasible point was found with eight of the 14 equality constraints still violated at termination of the program. The problem became much more difficult to solve by allowing the y_i to be unrestricted, which made the equality constraints more difficult to satisfy. The transformation $x_i = \sin^2 y_i$ was used in experiment 21 and the optimum point was found in the transformed problem in 2.76 times the reference time.

A careless error was made in experiment 22 using the transformation $x_i=e^{y}i$. Lower bounds were left on all 24 variables, but no compensation was made for the fact that negative values of y_i were necessary to start from the same starting point as in the other experiments. This error was corrected in experiment 23 by the same transformation but with the lower bounds removed, however, no feasible point was found and the program was making very slow progress. In experiment 24, the transformation $x_i=e^{y_i}/(e^{y_i} + e^{-y_i})$ was

attempted, but was unsuccessful in finding a point that would satisfy all constraints.

Two transformations were tried on problem 4. The reference run and optimum point is listed in experiment 25. The transformation $x_i=e^{y_i}$ yielded a slightly improved objective function in experiment 26 and a different optimum point, but also took slightly longer to solve. In experiment 27, the transformation $x_i=|y_i|$ was used and it was found to yield the same optimum as the reference run, and in about the same time. The transformation $x_i=e^{y_i}$ was also tried on problem 4 using the SUMT code (experiment 32) and provided the correct optimum, but in over five times the CPU time required by GRG for the same transformed problem.

Note from experiment 31 that SUMT still had not reached the optimum point in problem 20 after four minutes of CPU time. Thus none of the transformations considered here were tried on problem 20 using SUMT for two reasons:

(1) excessive computation time to reach an optimum solution,

(2) excessive preparation time to determine all the analytical first and second derivatives of the transformed problems.

From the preceding discussion, the following observations should be kept in mind when considering a transformation of variables using the nonlinear programming codes

GRG and SUMT:

(1) transformations generally make a nonlinear program harder to solve and can substantially increase the computer time required;

(2) transformations are less likely to cause difficulty when used in problems subject to finite lower or upper bounds;

(3) starting points and bounds on variables must be handled and adjusted carefully when preparing a transformed problem;

(4) when confronted with a real-world problem whose solution is not known in advance, it is always wise to try several different carefully chosen starting points to determine if the solution can be improved.

In view of these observations, it can be concluded that variable transformations have an adverse effect on nonlinear programming codes such as SUMT and GRG, and that they should not be attempted unless the codes have difficulty in reaching a solution using the original variables.

C. TRANSFORMATIONS OF VARIABLES TO OBTAIN SEPARABILITY

Once the analyst decides upon separable programming as a method for solution, he is faced with the problem of how to transform his model to separable form. This section discusses some of the transformations that can be used to eliminate interaction between variables.

Converting a nonlinear programming model into an approximate version with separable functions increases the size of the model in two ways. First, the separability transformation introduces new constraints and variables. Second, the subsequent linearization expands the number of constraints and variables even further.

The transformations described in this section are located in several references including Hadley [13], Beale [17] [18], Wagner [27], and McCormick [28]. No single reference provides a complete treatment of all the transformations discussed here.

1. Transformations for Product Terms and Exponential Expressions

Any product term of the form $x_i x_j$ appearing in a constraint or objective function can be eliminated by defining two new variables y_i and y_j as follows:

$$y_i = \frac{x_i + x_j}{2} \quad , \quad y_j = \frac{x_i - x_j}{2} \qquad (16)$$

Then $x_i x_j = y_i^2 - y_j^2$ which provides a separable form in the new variables. For every product term in the problem formulation, substitute $y_i^2 - y_j^2$, and add the two additional constraints defined in (16). Since y_j involves the difference of the original variables, it will be unrestricted in sign even if x_i and x_j are non-negative.

An alternative method to separate $x_i\ x_j$ is a log transformation which can be generalized to handle product terms of three or more variables. The original variables, however, must be strictly positive since $\ln 0 = -\infty$, and the logarithm of a negative argument is undefined. Setting $y_k = x_i\ x_j$ and taking the natural logarithm yields:

$$y_k = \ln x_i + \ln x_j \tag{17}$$

In the original problem formulation, each $x_i x_j$ term is replaced by y_k, and the additional constraint (17) is imposed. If the only nonseparable term in the problem is $x_i x_j$, introduction of the variable y_k and the additional constraint will result in a separable format.

To separate expressions of the form $\exp.\left(\sum_i h_i(x_i)\right)$, an example of which is $\exp.\ (ax_1^2 + bx_2)$, introduce the new variable y_k, and take the natural logarithm as follows:

$$\begin{aligned} y_k &= \exp.\ (ax_1^2 + bx_2) \\ \ln y_k &= ax_1^2 + bx_2 \end{aligned} \tag{18}$$

For expressions of the form $x_1\ x_j$, $x_i > 0$, $x_j \geq 0$, take the natural logarithm and proceed as follows:

$$\begin{aligned} v_j &= x_j + \epsilon \quad , \epsilon > 0 \\ z &= v_j \ln x_i \\ v_j &= y_i + y_j \\ \ln x_i &= y_i - y_j \end{aligned} \tag{19}$$

Therefore $z=v_j \ln x_i$ can be replaced in the formulation by:

$$z=y_i^2 - y_j^2 \qquad (20)$$

Himmelblau problem 16 has numerous cross product terms in both the objective functions and constraints. All cross product terms were replaced by transformations of the form $y_i^2-y_j^2$. The effect of these transformations was to increase both the dimensionality and degree of difficulty of problem. Refer to experiments 33 and 34 in Appendix B. The original problem had nine variables and 13 nonlinear inequality constraints. The transformed problem had 41 variables, 13 nonlinear inequality constraints, and an additional 32 equality constraints. The two experiments differed only in a parameter tolerance; however, each constructed the same local optimum point. Therefore, when using GRG, the analyst should avoid making product transformations because of the additional complexity entailed. However, if only a linear programming package with separable option is available, the analyst must make these transformations.

2. Transformation of Quadratic Expressions Into Diagonal Form

Every quadratic form can be expressed in terms of a symmetric matrix Q associated with its coefficients. The quadratic programming problem can be formulated in

matrix notation as follows:

$$\begin{aligned} &\text{minimize} && \underline{x}^T \underline{Q}\, \underline{x} \\ &\text{subject to:} && \underline{A}\, \underline{x} \leq \underline{b} \\ & && \underline{x} \geq 0 \end{aligned} \tag{21}$$

A quadratic form is defined to be positive definite if $\underline{x}^T \underline{Q}\, \underline{x}$ is strictly positive for all $\underline{x} \neq 0$; it is defined to be positive semi-definite if $\underline{x}^T \underline{Q}\, \underline{x}$ is non-negative for all $\underline{x}$.

By a suitable change of variables

$$\underline{x} = \underline{R}\, \underline{y} \tag{22}$$

the quadratic problem can be transformed to

$$\begin{aligned} &\text{minimize} && \underline{y}^T \underline{R}^T \underline{Q}\ \underline{R}\, \underline{y} \\ &\text{subject to:} && \underline{A}\, \underline{R}\, \underline{y} \leq \underline{b} \\ & && \underline{R}\, \underline{y} \geq \underline{0} \end{aligned} \tag{23}$$

If an $\underline{R}$ matrix can be found that will make $\underline{R}^T \underline{Q}\, \underline{R}$ diagonal, then this will make $\underline{y}^T \underline{R}^T \underline{Q}\, \underline{R}\, \underline{y}$ a sum of squares and therefore separable. It is possible to diagonalize any symmetric, nonsingular matrix without the laborious effort entailed by the Gram-Schmidt orthogonalization process in solving for eigenvalues (see Ref. /29/). For a symmetric matrix $\underline{Q}$, a sequence of elementary row operations followed by the same sequence of elementary column operations will diagonalize the matrix. The same sequence of operations when applied to the identity matrix will yield

a matrix $\underline{R}^T$, such that $\underline{R}^T\ \underline{Q}\ \underline{R}$ is a diagonal matrix.

To illustrate this point, consider the following symmetric matrix.

$$\underline{Q} = \begin{pmatrix} 1 & 2 & -3 \\ 2 & 5 & -4 \\ -3 & -4 & 8 \end{pmatrix} \qquad (24)$$

The first step is to augment it with the identity matrix:

$$(\underline{Q},\underline{I}) = \left(\begin{array}{ccc|ccc} 1 & 2 & -3 & 1 & 0 & 0 \\ 2 & 5 & -4 & 0 & 1 & 0 \\ -3 & -4 & 8 & 0 & 0 & 1 \end{array}\right) \qquad (25)$$

Step 2: pivot on row 1, to yield:

$$\left(\begin{array}{ccc|ccc} 1 & 2 & -3 & 1 & 0 & 0 \\ 0 & 1 & 2 & -2 & 1 & 0 \\ 0 & 2 & -1 & 3 & 0 & 1 \end{array}\right)$$

Step 3: pivot on column 2, to yield:

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & -2 & 1 & 0 \\ 0 & 2 & -1 & 3 & 0 & 1 \end{array}\right)$$

Step 4: pivot on row 2, to yield:

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & -2 & 1 & 0 \\ 0 & 0 & -5 & 7 & -2 & 1 \end{array}\right)$$

Step 5: pivot on column 2, to yield a diagonal matrix

$$(\underline{R}^T \underline{Q}\ \underline{R},\ \underline{R}^T) = \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -2 & 1 & 0 \\ 0 & 0 & -5 & 7 & -2 & 1 \end{array}\right) \quad (26)$$

The actual computation involved Gauss-Jordan elimination pivoting on row 1, column 1, row 2, column 2 in order, so as to reduce all off diagonal elements to zero. These simple operations can be programmed easily. A program was written for this study to determine the amount of computation time required to diagonalize matrices of different sizes. The results of the experiments are presented in the next subsection.

Complications arise whenever there is a zero element along the diagonal in which case the Gauss-Jordan reduction scheme breaks down. The algorithm can be made to handle the case of an arbitrary zero element along the diagonal, or the case in which all diagonal elements are zero. In the first case, the matrix can still be diagonalized by interchanging the row in which it appears with the next lower row in which the diagonal element is non-zero. This is followed by interchanging the corresponding two columns. This has the

effect of moving the zero element down the diagonal to the next lower position, allowing normal pivot operations to be continued. It should be observed that each iteration of the algorithm can introduce a zero in those diagonal elements below the current pivot row. Whenever a zero element is encountered, this procedure is repeated.

If all diagonal elements are zero, choose i,j such that $a_{ij} \neq 0$, and apply the row operation $R_i \rightarrow R_j + R_i$, and the corresponding column operation $C_i \rightarrow C_j + C_i$ (where $\rightarrow$ means "is replaced by"). This has the effect of bringing $2a_{ij}$ into the a_{ii} diagonal position. This element can then be moved to the first diagonal position by another interchange of row and column. An $a_{ij} \neq 0$ must exist because the $\underline{Q}$ matrix is required to be nonsingular. The $\underline{Q}$ matrix will be in the following form:

$$\underline{Q} = \left(\begin{array}{c|c} a_{11} & a_{ij} \\ \hline a_{ij} & \underline{B} \end{array}\right) \qquad (27)$$

Here $\underline{B}$ is a symmetric matrix of order less than $\underline{Q}$, still having zeros along its diagonal. The diagonalization algorithm is then used to zero out the elements in the first row and column. This process is then repeated until, by induction, $\underline{Q}$ is brought into diagonal form.

There are several observations to be made regarding this transformation. First, the matrix $\underline{R}^T$ will be lower triangular only if no zero elements appear along the diagonal during execution of the algorithm. When this is the case, $\underline{R}$ will be upper triangular. Second, the diagonal form resulting from this algorithm does not yield the eigenvalues of the matrix $\underline{Q}$, but rather a simple way of transforming a quadratic form into separable form. Third, the algorithm must include a test at each iteration to determine if the diagonal element in the pivor row is equal to or very close to zero.

3. Experiments in the Diagonalization of Quadratic Forms

Table 3 lists the experiments that were conducted using the diagonalization algorithm described in the previous subsection. Two types of matrices were diagonalized. The first type was tridiagonal in which the main diagonal and the adjacent diagonals all had non-zero integer elements from the interval range (-50,50). The entries were determined by a random number generator. The second type was a random entry matrix in which the random number generator was used to determine not only the magnitude of the element, but also its location in the matrix. For both types of matrices, the size of matrices tested were 20x20, 40x40, 60x60, and 80x80. Each random entry matrix size tested

TABLE 3: Application of a Diagonalization Algorithm

A. Tridiagonal Matrix

Matrix Size	Nonzero Elements	Time to Diag. (sec)	Total CPU Time (sec)
20x20	58	0.12	0.94
40x40	118	1.12	3.65
60x60	178	3.93	9.42
80x80	238	9.24	18.58

B. Random Entry Matrix

Matrix Size	Class	Nonzero Elements	Time to Diag. (sec)	Total CPU Time (sec)
20x20	Sparse	40	0.15	0.98
	Medium	200	0.13	1.01
	Dense	360	0.13	1.14
40x40	Sparse	160	1.12	3.61
	Medium	800	1.12	4.03
	Dense	1440	1.22	4.93
60x60	Sparse	360	4.05	9.97
	Medium	1800	3.95	10.66
	Dense	3240	3.86	11.64
80x80	Sparse	640	9.49	18.96
	Medium	3200	9.80	21.55
	Dense	5760	9.26	22.72

was further classified and tested as sparse, medium, or dense, depending on the number of non-zero elements. The classification sparse, medium, and dense was used to describe matrices having, respectively, 10%, 50%, and 90% of its elements non-zero. A timer routine was used to compute the actual time spent in diagonalization, since the larger matrices took a proportionately greater time in generating matrix elements.

The computational algorithm used did not take advantage of the degree of sparseness of the test matrices, so the diagonalization time is essentially the same for a given matrix size. The time did not vary in direct proportion to the number of matrix elements but instead increased more rapidly as the size increased. In fact, when the number of rows (and columns) is increased by a factor of k, the time can be expected to increase by a factor of k^3. For example the 80x80 matrix was 1.33 times as large as the 60x60 matrix, but the computational time increased by a factor of 2.41.

The analyst must be aware of the time and preparation necessary to convert a quadratic expression into diagonal form, and must weigh this against whether to apply a series of product type transformations in order to separate the variables.

To conclude this section, the analyst must keep in mind that transformations to separable form can greatly increase the size of the model and thus make it less economical to solve. The analyst should also be aware that the next step, in the conversion of the separable form to piecewise linear approximations of the functions involved, expands the model further by introducing special sets of variables.

D. TRANSFORMATION TO UNCONSTRAINED PROBLEMS BY USE OF PENALTY AND BARRIER FUNCTIONS

A general nonlinear programming problem can be reduced to a sequence of unconstrained optimization problems by a transformation which combines the objective function and constraints. The minima of the new unconstrained function approximate the solution to the constrained problem. Exterior point techniques compute a sequence of points generally outside the feasible region of the original problem. This is accomplished by addition of a penalty term to the objective function that is a function of only the violated constraints. A useful penalty function for inequality constraints of the form $g_i(\underline{x}) \geq 0$ is:

$$P_j(\underline{x}) = \min (0, g_j(\underline{x}))^2 \qquad (28)$$

The optimization problem becomes the minimization of:

$$f(\underline{x}) + r_k \sum_{j=1}^{m} P_j(\underline{x}) \qquad (29)$$

where r_k is an appropriate weight. In the limit as r_k becomes large, constrained minima of the original problem are approached by unconstrained minima of the transformed problem.

An important attribute of the penalty-function approach is that the initial search point does not have to satisfy the constraints. A simultaneous solution to the constraint equations is attained concurrently to the attainment of constrained relative minima. A major

disadvantage of exterior point transformations is that the transformed problem becomes progressively ill-conditioned as the penalty function increases. In addition, the numerical errors in the penalty terms become significant for large penalty coefficients. Because of these difficulties, it may be very difficult to satisfy the constraints with any desired accuracy.

Interior point techniques compute a sequence of feasible solutions to the original problem. The inside penalty function establishes a barrier within the feasible region which can not be crossed by a search for the constrained minimum. This transformation prevents the solutions from violating the inequality constraints which are gradually approached as the barrier is relaxed. Useful barrier functions are $\langle \bar{g}_j(x) \rangle^{-1}$ and $\ln(\langle \bar{g}_j(x) \rangle^{-1})$. The optimization problem becomes:

$$\text{minimize} \quad f(\underline{x}) + r_k \sum_{j=1}^{m} B_j(\underline{x}) \qquad (30)$$

where r_k is an appropriate weight and there are m such inequality constraints.

The starting point for interior point techniques must be a point which strictly satisfies the constraints. After a minimum has been obtained for one value of r_k, a new and smaller r_k is used in the next search. Each constrained relative minimum of $f(\underline{x})$ is approached asymptotically by a relative minimum of (30).

Interior point methods are incapable of handling equality constraints and are combined with a penalty term to counter this difficulty. The sequential unconstrained minimization technique (SUMT) of Fiacco and McCormick /3/ is a mixed interior-exterior penalty function technique. For the general nonlinear programming problem:

minimize $f(\underline{x})$

subject to:

$$g_j(\underline{x}) \geq 0 \qquad j = 1,2,\ldots,m_1$$
$$h_j(\underline{x}) = 0 \qquad j = m_1+1,\ldots,m \tag{31}$$

the SUMT code minimizes the unconstrained penalty function transformation:

$$P(\underline{x},r_k) = f(\underline{x}) - r_k \sum_{j=1}^{m} \ln g_j(\underline{x}) + \frac{1}{r_k} \sum_{j=m_1+1}^{m} h_j(\underline{x})^2 \tag{32}$$

The objective function and inequality constraints can be nonlinear functions of the variables but the equality constraints must be linear functions of the variables in order to guarantee convergence to the solution of the nonlinear programming problem.

There are several disadvantages associated with the mixed interior-exterior point methods. The first is that the Hessian matrix of the $P(\underline{x},r_k)$ function becomes progressively more ill-conditioned as the minimum is approached, so search directions may be misleading. Second, the rate of convergence

is slowed considerably as the structure of the unconstrained problem becomes more unfavorable. The biggest disadvantage from the user's standpoint is the amount of preparation time required to compute the analytical first and second derivatives of the original objective function and constraints. Unless most of the derivatives are zero or constant, this will not only require a lot of time but is also more prone to human error.

References /30/-/33/ give good discussions of barrier and penalty function techniques. The excellent treatment of unconstrained nonlinear programming techniques makes Ref. /30/ particularly useful as a general reference.

Although it is not the intent of this thesis to compare the efficiency of the SUMT and GRG codes, it should be noted that the SUMT method generally required a significantly greater amount of computation time than did the GRG method. As pointed out in various sections of this chapter and the next one, GRG is a very versatile code in that scaling and transformations can be applied easily by modifying the user supplied subroutine GCOMP. On the other hand, SUMT requires computation of analytical gradient and Hessian functions for the GRAD1 and MATRIX subroutines. This can require a substantial amount of preparation and debugging time. However, SUMT does have an option that

enables it to compute numerical approximations for the gradient and Hessian functions by finite differencing.

V. THE USE OF SCALING, ROTATION, AND TRANSLATION IN CONSTRAINED OPTIMIZATION PROBLEMS

A. USE OF SCALING IN NONLINEAR PROGRAMMING PROBLEMS

The objective of this chapter is to determine how sensitive the GRG code is to scaling in nonlinear programming problems. The analyst is responsible for scaling his problem and very real difficulties can be encountered if he attempts to solve a problem using GRG without making an effort to scale it first. Commercial linear programming codes are forgiving in the sense that the code will perform row and column scaling operations on the problem tableau. This is not true of GRG.

Although scaling is important in both linear and nonlinear programming problems, it is especially critical in nonlinear programming. Attempting to solve a linear programming problem without scaling is to run the risk of introducing round-off errors which alter the original problem and may even result in a false optimum being designated. However, in the nonlinear programming problem, if the problem is poorly scaled, there is a good possibility the nonlinear programming code used will be unable to even find a feasible solution, let alone provide an optimal solution.

The efficiency and rate of convergence of optimization methods depends very critically on the given function f(x) and the scales used for the variables. A great deal of caution must be used with even the simplest scale changes, such as $x_i = a_i y_i$, $a_i > 0$, since many important aspects of the optimization algorithm will not be left invariant.

To illustrate some points consider the following problem:

$$\begin{aligned} &\text{Minimize} \quad f(\underline{x}) = x_1^2 + x_2^2 \\ &\qquad x_1, x_2 \geq 0 \\ &\qquad \underline{x}^0 = (1,1)^T \end{aligned} \tag{33}$$

The function isocontours are concentric circles in this case. The method of steepest descent gives the descent direction $\underline{d} = -\nabla f(1,1) = (-2,-2)^T$, and the solution $\underline{x}^* = (0,0)^T$ in one step. By introducing new variables:

$$\begin{aligned} y_1 &= 0.1\, x_1 \\ y_2 &= x_2 \end{aligned} \tag{34}$$

the efficiency of steepest descent is radically changed.

The isocontours of the new objective function

$$f(\underline{y}) = 100\, y_1^2 + y_2^2 \tag{35}$$

form a deep and narrow valley along the y_2 coordinate, and the gradient vector calculated at the same point as before, $\underline{d} = -\nabla f(.1,1) = (-20,2)^T$ makes an angle of nearly 90

degrees with the y_2 axis. Thus steepest descent is inefficient in handling the rescaled problem.

Second order Newton methods using the Hessian matrix do not have this drawback, and may be considered invariant with respect to linear changes of scale of the variables. The direction vector is reoriented towards the solution in the scaled system of variables. For the example of this section:

$$\underline{d} = - \text{Hess } (.1,1)^{-1} \nabla f(.1,1) = - \begin{bmatrix} .005 & 0 \\ 0 & .5 \end{bmatrix} \begin{bmatrix} 20 \\ 2 \end{bmatrix}$$

$$\underline{d} = - (.1,1)^T \qquad (36)$$

When $f(\underline{x})$ is a very complicated function of its variables, it may be very difficult to scale the problem. Intuitively, in general constrained nonlinear optimization problems, "well-scaled" problems are those in which similar changes in the variables lead to similar changes in the objective function.

For a quadratic function of n variables, it is possible to diagonalize the quadratic form $\underline{Q}$, as suggested in the preceding chapter, and then scale the diagonal elements of the $\underline{R}^T\underline{Q}\ \underline{R}$ matrix so that they are approximately equal. The next section presents and discusses detailed experiments that were done on two well-scaled problems using the GRG nonlinear programming code.

The problem of scaling has another interesting aspect. In a simple two-dimensional problem, if $10^{-5} \leq x_1 \leq 10^{-2}$ and $10^{-2} \leq x_2 \leq 10^{3}$, the feasible region has the shape of a very narrow band. It would be very difficult to search for a minimum in such a band because the step length would have to be very short to avoid violation of the constraint imposed by x_1.

Difficulties are also encountered by performing arithmetic operations with numbers of different orders of magnitude. To scale the variables so that they are of the same order of magnitude, where $\ell_i \leq x_i \leq u_i$, introduce a new variable y_i defined as follows:

$$y_i = 2 \frac{a_i x_i - a_i (\ell_i + u_i)}{u_i - \ell_i} \qquad (37)$$

where y_i lies in the interval $(-a_i, a_i)$. If all a_i are taken equal to one, then all variables would be located within a hypercube of length 2 about the origin of the new coordinate system.

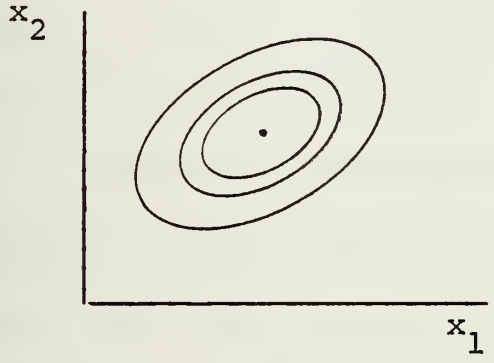
References [34] and [35] contain descriptions of self-scaling algorithms for unconstrained minimization problems.

B. COMPUTATIONAL EXPERIENCE WITH SCALING, TRANSLATION, AND ROTATION OPERATIONS ON NONLINEAR PROGRAMMING PROBLEMS

The effect of translation, rotation, and scaling

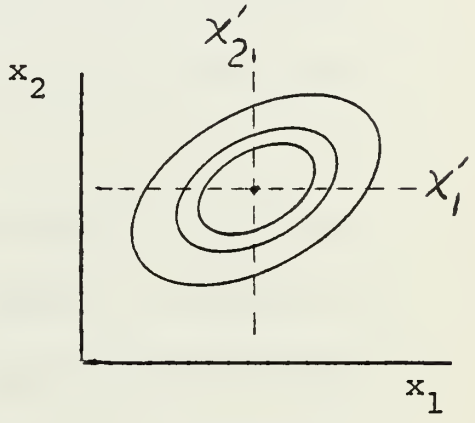
operations can be illustrated by the following sketches of a two-dimensional quadratic function.

Figure 2: Effect of translation, rotation, and scaling operations on a quadratic function



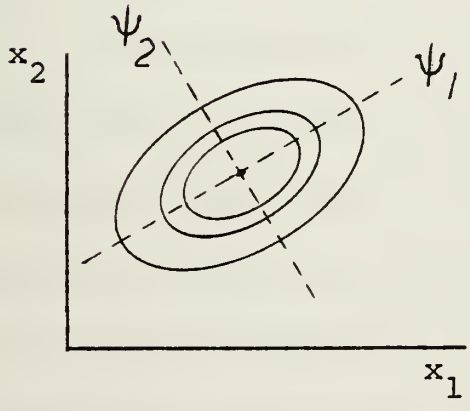
Original Problem

(A)



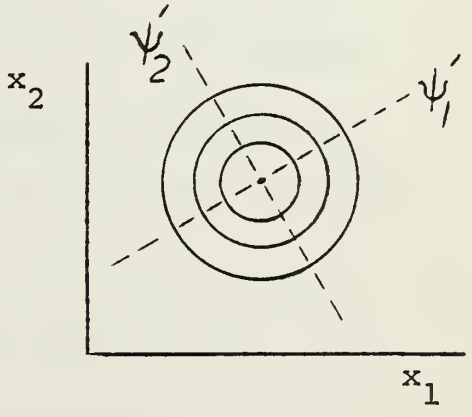
Translation

(B)



Rotation

(C)



Scaling

(D)

The original problem is transformed to a new coordinate system by translation of the old origin to the minimum of the function. The axes are then rotated to achieve symmetry of the function contours with respect to the new coordinate system. Finally, the coordinates are scaled to make the function contours circular.

Appendix B enumerates the experiments of scaling, translation, and rotation that were applied to problem 16 and 20 using the GRG code. All the experiments were conducted using GRG because the user supplied subroutines could be easily modified by the simple addition of a few Fortran statements, and the appropriate adjustment of the input vector.

The experiments on scaling were done in a reverse sense, that is, given a fine-tuned properly scaled test problem to begin with, at what point would different scale magnitudes affect the ability of GRG to find a solution. Refer again to Appendix C for a description of GRG. The authors of that code feel that proper scaling of variables and functions is critical to success of the code. They further recommend that constraints be scaled to have absolute values below 100. There are no printouts of the gradients of the constraints and objective function, but the user should suspect scaling problems if there are

large values in the reduced gradient array which prints out with final solution information. These values should all be approximately zero when the program terminates.

The experiments were run by reading in an initial vector $\underline{y}$, multiplying the components of $\underline{y}$ by appropriate scaling factors to get a new vector $\underline{x}$ which was used to evaluate the constraints and objective function. In successive iterations the code tries to optimize the $\underline{y}$ vector, and in so doing, should construct the equivalent optimum $\underline{x}$ vector. The starting point was modified so that the same starting $\underline{x}$ vector is used in the functional evaluations. There was no way to use constraint values to predict scaling difficulties in these experiments. The test problems were adjusted to account for scaling factors in such a manner that the same starting point was always used. Consequently, the initial constraint values were always the same.

In experiments 35 and 38 on problem 16, the initial components of the $\underline{y}$ vector had the value 0.01,0.1, or 1. Both experiments ran to the correct optimum. Likewise, experiment 39 resulted in the correct optimum with the initial components of $\underline{y}$ each being 0.1. In experiment 36, the $\underline{y}$ vector initially had components of .0001 or 1, causing the initial step size to be .8 (10^{-7}), which was

quickly reduced to zero in an attempt to satisfy the constraints. This caused the program to terminate. Experiment 37 produced the same result. The initial $\underline{y}$ components were either .001, .01, or 1; the initial step size was (10^{-6}) which was quickly reduced to zero causing program termination. It should be noted in the experiments where the scaling factors did not prevent GRG from finding the optimum point, the time required to solve the problem increased over the reference time of 2.90 seconds.

Problem 20 proved to be a much harder problem to solve when using scale factors. Its sensitivity is probably due to the equality constraints, and the ratios of one variable to a weighted sum of several variables appearing in most of the constraints. In experiments 49 through 52, although only moderate scale factors were used, each program eventually terminated without finding a feasible point when the step size was reduced to zero. In each of these experiments, what was most noticeable was the difference in magnitude of the components of the reduced gradient. The largest difference occurred when all variables were scaled by the same factor. The scale factor of 10 produced reduced gradient elements as large as 10^6 while terms as large as 10^5 occurred when the scale factor was 2. At the optimum point, all elements of the reduced gradient should

equal zero. Only in experiment 53, where a scale reduction was tried, was GRG successful in reaching the optimum point, taking about 1 second longer than the reference run.

Translation of coordinates was tried in experiment 40 on problem 16 and in experiment 54 on problem 20. The simple translation made was:

$$x_i = y_i + B_i \qquad i = 1,\ldots,n \qquad (38)$$

where y_i is the i^{th} component of the input $\underline{y}$ vector, and x_i is the i^{th} component of the vector used in the functional evaluations. Both problems were solved easily, each taking approximately one second longer to solve than the corresponding reference run. Translation did not appear to be important in these experiments. A case where it might be of use is in a goal programming model formulated for use with GRG in which each decision variable has the same goal.

Finally, a number of pairwise rotation experiments were tried on problems 16 and 20. These comprise experiments 41 through 48, and experiments 55 through 64. Single pair rotations were of the form:

$$\begin{aligned} x_i &= y_i - y_j \\ x_j &= y_i + y_j \end{aligned} \qquad (39)$$

where, as previously, y_i is a component of the input vector and x_i is a component of the vector used in evaluating

constraints and the objective function. All of these single pair rotations worked although the global optimum was not always reached. In general, experiments on problem 16 took about 0.5 seconds, and experiments on problem 20 about 5.0 seconds longer than the corresponding reference runs.

In experiment 47, a multiple pair rotation was tried on problem 16 and was successful in reaching the global optimum. The final experiment on problem 16 was experiment 48 in which the input vector $\underline{y}$ was premultiplied by an arbitrary matrix consisting of + 1's, -1's, and 0's. A Gauss-Jordan reduction program was used to determine the initial $\underline{y}$ vector to provide the same starting $\underline{x}$ vector. Again, GRG solved the problem quickly with the correct optimum.

Experiments 60 to 62 were multiple pairwise rotations of the $\underline{y}$ coordinates, but each failed to provide a feasible point. The probable cause of failure was the difficulty of satisfying the 12 nonlinear equality constraints. Two or more of these constraints were violated in each experiment. Experiment 63 was another multiple pair rotation, in which 4 of the single pair combinations from previous good experiments was tried. This experiment resulted in a non-optimal solution. The final experiment tried was pairwise rotation of 12 pairs of variables

(treated consecutively), but it too failed to determine a feasible point.

To sum up the results of the experiments in this section, it can be stated that scaling is a significant factor, affecting the ability of GRG to find the feasible region, and then to optimize the function successfully. It is hard to give guidelines for scaling a constrained non-linear program, but a first step would be to convert the coefficients in the objective function and constraints to the same order of magnitude. If the code fails to find a feasible point, look at the values of the constraints to see if any exceed the recommended figure of $\pm$ 100. Finally, check the reduced gradient in the solution information, to see if there are any large components. At the optimum point, these values should be essentially zero.

The experiments with translation and rotation operations indicate that these operations make it harder for the GRG code to find the optimum point, and thus are not recommended.

APPENDIX A

Test Problems Used with GRG and SUMT Codes

A. HIMMELBLAU PROBLEM 16

Source: J.D. Pearson, On Variable Metric Methods of Minimization, Research Analysis Corporation Report. RAC-TP-302, MCLean, Virginia, May 1968.

Number of variables: 9

Number of constraints: 13 nonlinear inequality constraints
1 upper bound

Objective function:

Maximize: $f(\underline{x}) = 0.5(x_1x_4 - x_2x_3 + x_3x_9 - x_5x_9 + x_5x_8 - x_6x_7)$

Constraints:

$$
\begin{aligned}
1 - x_3^2 - x_4^2 &\geq 0 \\
1 - x_9^2 &\geq 0 \\
1 - x_5^2 - x_6^2 &\geq 0 \\
1 - x_1^2 - (x_2 - x_9)^2 &\geq 0 \\
1 - (x_1 - x_5)^2 - (x_2 - x_6)^2 &\geq 0 \\
1 - (x_1 - x_7)^2 - (x_2 - x_8)^2 &\geq 0 \\
1 - (x_3 - x_5)^2 - (x_4 - x_6)^2 &\geq 0 \\
1 - (x_3 - x_7)^2 - (x_4 - x_8)^2 &\geq 0 \\
1 - x_7^2 - (x_8 - x_9)^2 &\geq 0 \\
x_1x_4 - x_2x_3 &\geq 0 \\
x_3x_9 &\geq 0 \\
-x_5x_9 &\geq 0 \\
x_5x_8 - x_6x_7 &\geq 0 \\
x_9 &\geq 0
\end{aligned}
$$

Starting point:

$$x_i = 1 \qquad i = 1,.....,9$$

$$f(\underline{x}) = 0$$

Optimum point:

$$\underline{x}^* = (0.9971, -0.0758, 0.5530, 0.8331, 0.9981, -0.0623, 0.5642, 0.8256, 0.0000024)^T$$

$$f(\underline{x}^*) = 0.8660$$

B. HIMMELBLAU PROBLEM 4

Source: J. Bracken and G.P. McCormick, "Selected Applications of Nonlinear Programming," John Wiley and Sons, Inc., New York, 1968.

Number of variables: 10

Number of constraints: 3 linear equality constraints

10 bounds on independent variables

Objective function:

$$\text{Minimize: } f(\underline{x}) = \sum_{i=1}^{10} x_i \left(c_i + \ln \left(\frac{x_i}{\sum_{j=1}^{10} x_j} \right) \right)$$

Constraints:

$$h_1(\underline{x}) = x_1 + 2x_2 + 2x_3 + x_6 + x_{10} - 2 = 0$$

$$h_2(\underline{x}) = x_4 + 2x_5 + x_6 + x_7 - 1 = 0$$

$$h_3(\underline{x}) = x_3 + x_7 + x_8 + 2x_9 + x_{10} - 1 = 0$$

$$x_i \geq 0 \qquad i = 1,.....,10$$

Starting Point: $x_i = 0.1 \qquad i = 1,.....,10$

$$f(\underline{x}) = -20.961$$

Optimum Point:

$\underline{x}^*$=(0.0406,0.1477,0.7832,0.0014,0.4853,0.0007
0.0274,0.0180,0.0375,0.0969)

$f(\underline{x}^*)$= - 47.761

Data:

c_1=-6.089 c_2=-17.164 c_3=-34.054 c_4=-5.914

c_5=-24.721 c_6=-14.986 c_7=-24.100 c_8=-10.708

c_9=-26.662 c_{10}=-22.179

C. HIMMELBLAU PROBLEM 20

Source: D.A. Paviani Ph.D dissertation, The University of Texas, Austin, Texas, 1969

Number of variables: 24

Number of constraints: 12 nonlinear equality constraints

2 linear equality constraints

6 nonlinear inequality constraints

24 bounds on independent variables

Objective function:

Minimize $f(\underline{x}) = \sum_{i=1}^{24} a_i x_i$

Constraints:

$$h_i(x) = \frac{x_{(i+12)}}{b_{(i+12)} \sum_{j=13}^{24} \frac{x_j}{b_j}} - \frac{c_i x_i}{40 b_i \sum_{j=i}^{12} \frac{x_j}{b_j}} = 0 \quad i=1,\ldots,12$$

$$h_{13}(x) = \sum_{i=1}^{24} x_i - 1 = 0$$

$$h_{14}(x) = \sum_{i=1}^{12} \frac{x_i}{d_i} + f \sum_{i=13}^{24} \frac{x_i}{b_i} - 1.671 = 0$$

where $f=(0.7302(530) \cdot \left(\frac{14.7}{40}\right)$

$$- \frac{\left[x_i + x_{(i+12)}\right]}{\sum_{j=1}^{24} x_j} + e_i \geq 0 \qquad i=1,2,3$$

$$- \frac{\left[x_{(i+3)} + x_{(i+15)}\right]}{\sum_{j=1}^{24} x_j} + e_i \geq 0 \qquad i=4,5,6$$

$$x_i \geq 0 \quad i=1,....,24$$

Starting Point: $x_i=0.04$ $i=1,....,24$

$f(\underline{x}) = 0.14696$

Optimum Point:

$\underline{x}^* =$ (0.0,0.1072,0.1114,0.0,0.0,0.0,0.0755,0.0,
0.0,0.0,0.0,0.0112,0.0,0.1928,0.2886,0.0,
0.0,0.0,0.2129,0.0,0.0,0.0,0.0,0.0004)

$f(\underline{x}^*)$ 0.055658041

Data:

i	a_i	b_i	c_i	d_i	e_i
1	0.0693	44.094	123.7	31.244	0.1
2	0.0577	58.12	31.7	36.12	0.3
3	0.05	58.12	45.7	34.784	0.4
4	0.20	137.4	14.7	92.7	0.3
5	0.26	120.9	84.7	82.7	0.6

Data: Cont'd.

i	a_i	b_i	c_i	d_i	e_i
6	0.55	170.9	27.7	91.6	0.3
7	0.06	62.501	49.7	56.708	
8	0.10	84.94	7.1	82.7	
9	0.12	133.425	2.1	80.8	
10	0.18	82.507	17.7	64.517	
11	0.10	46.07	0.85	49.4	
12	0.09	60.097	0.64	49.1	
13	0.0693	44.094			
14	0.0577	58.12			
15	0.05	58.12			
16	0.20	137.4			
17	0.26	120.9			
18	0.55	170.9			
19	0.06	62.501			
20	0.10	84.94			
21	0.12	133.425			
22	0.18	82.507			
23	0.10	46.07			
24	0.09	60.097			

APPENDIX B

Experiments Conducted with SUMT and GRG Codes

A. VARIABLE TRANSFORMATIONS USING GRG-HIMMELBLAU PROBLEM 16

Experiment Number 1

$f(\underline{x}^*)$= .43305879 iterations=8 CPU=1.71 sec.

$\underline{x}^*$= (.8660,.4999,.8660,.4999,0.0,0.0,.8660,1.5000,1.0)

Experiment Number 2

TOL parameter changed in DEGEN subroutine to 10^{-4}

$f(\underline{x}^*)$= .86603619 iterations=17 CPU=2.90 sec.

$\underline{x}^*$=(0.0,0.0,.8660,-.5,0.0,-1.0,.8660,.5,1.0)

Experiment Number 3

Lower bounds placed on all variables

$f(\underline{x}^*)$= .50000076 iterations=10 CPU=2.18 sec.

$\underline{x}^*$=(.8660,.5000,.5006,.8657,0.0,0.0, .8660,1.5,1.0)

Experiment Number 4

Lower bounds on all variables

Transformation: $x_i=y_i^2$

$f(\underline{x}^*)$= .50001561 iterations=20 CPU=3.56 sec.

$\underline{x}^*$=(1.0,.9948,0.0,1.0,0.0,1.0,0.0,.9948,1.0)

Experiment Number 5

Run experiment number 3 from optimum generated by SUMT

$f(\underline{x}^*)$= .86602621 iterations=11 CPU=1.98 sec.

$\underline{x}^*$= (.9306,.7071,.0254,1.0,.9308,.7067,0.0,1.0,0.0)

Experiment Number 6

Run experiment number 4 from optimum generated by SUMT

$f(\underline{x}^*)$= .86601997 iterations=5 CPU=1.38 sec.

$\underline{x}^*$= (.8660,.5000,0.0,1.0,.8660.,5000,0.0,1.0,0.0)

Experiment Number 7

Transformation: $x_i = |y_i|$, $y_i \geq 0$

$f(\underline{x}^*)$= .50000076 iterations=10 CPU=2.33 sec.

$\underline{x}^*$= (.8660,.5000,.5000,.8657,0.0,0.0,8660,1.5,1.0)

Experiment Number 8

Transformation: $x_i = |y_i|$; no lower bounds

$f(\underline{x}^*)$= .43301447 iterations=12 CPU=2.47 sec.

$\underline{x}^*$= (.8660,.5000, .8660,.5000, 0.0,0.0, .8660,1.5,1.0)

Experiment Number 9

Transformation: $x_i = e^{y_i}$; no lower bounds

$f(\underline{x}^*)$= .4833425 iterations=22 CPU=5.77 sec.

$\underline{x}^*$= (.9845,.2034,.08,.9968,.0215,.0613,.3403,.9682,.0279)

Experiment Number 10

Transformation: $x_i = e^{y_i}$, $y_i \geq 0$

No feasible point found

Experiment Number 11

Transformation: $x_i = \sin^2 y_i$; no lower bounds

No feasible point found

Experiment Number 12

Transformation: $x_i = \sin^2 y_i$; $y_i \geq 0$

No feasible point found

Experiment Number 13

Transformation: $x_i = \sin^2 y_i$; no lower bounds

$\underline{y}^0 = (.7854, .7854, \ldots\ldots, .7854)$

f(x*)= .50035409 iterations=12 CPU=2.59 sec.

$\underline{x}^* = (.9898, .1548, .1485, .9889, .0017, .0008, .2533, .8311, .2971)$

Experiment Number 14

Transformation: $x_i = |y_i|$; no lower bounds

Start from reported optimum (see Appendix A)

No feasible point found

Experiment Number 15

Transformation: $x_i = \dfrac{e^{y_i}}{e^{y_i} + e^{-y_i}}$; no lower bounds

No feasible point found

B. VARIABLE TRANSFORMATIONS USING GRG-HIMMELBLAU PROBLEM 20

Experiment Number 16

f($\underline{x}$*)= .055658041 iterations=30 CPU=13.56 sec.

$\underline{x}^* = (0.0, .1072, .1114, 0.0, 0.0, 0.0, .0755, 0.0, 0.0, 0.0, 0.0, .0112, 0.0, .1928, .2886, 0.0, 0.0, 0.0, .2129, 0.0, 0.0, 0.0, 0.0, .0004)$

Experiment Number 17

Lower bounds changed to inequality constraints

$f(\underline{x}^*)$=.055658019 iterations=28 CPU=23.03 sec.

Experiment Number 18

Transformation: $x_i=y_i^2$; $y_i \geq 0$

$f(\underline{x}^*)$=.055668194 iterations=47 CPU=28.25 sec.

Experiment Number 19

Transformation: $x_i=|y_i|$; $y_i \geq 0$

$f(\underline{x}^*)$=.05565887 iterations=70 CPU=40.11 sec.

Experiment Number 20

Transformation: $x_i=|y_i|$; no lower bounds

No feasible point found

Experiment Number 21

Transformation: $x_i=\sin^2 y_i$

$f(\underline{x}^*)$= .055672184 iterations=51 CPU=37.40 sec.

Experiment Number 22

Transformation: $x_i=e^{y_i}$, $y_i \geq 0$

$\underline{y}^0$= (-3.281,.....,-3.281)

Did not run because $\underline{y}^0$ violated lower bounds

Experiment Number 23

Transformation: $x_i=e^{y_i}$, no lower bounds

No feasible point found

Experiment Number 24

Transformation: $x_i = \dfrac{e^{y_i}}{e^{y_i} + e^{-y_i}}$; no lower bounds

No feasible point found

C. VARIABLE TRANSFORMATIONS USING GRG CODE-HIMMELBLAU PROBLEM 4

Experiment Number 25

$f(\underline{x}^*)$= -47.606887 iterations=9 CPU=1.68 sec.

$\underline{x}^*$=(.1278,.1678,.6454,.0033,.4838,.0018,.0273,.0303,.0265,.2439)

Experiment Number 26

Transformation: $x_i = e^{y_i}$

$f(\underline{x}^*)$=-47.751577 iterations=12 CPU=2.62 sec.

$\underline{x}^*$=(.0270,.1465,.7820,.0038,.4854,.0035,.0219,.0159,.0339,.1124)

Experiment Number 27

Transformation: $x_i = |y_i|$, $y_i \geq 0$

$f(\underline{x}^*)$= 47.606887 iterations=9 CPU=1.59 sec.

$\underline{x}^*$=(.1278,.1678,.6454,.0033,.4838,.0018,.0273,.0303,.0265,.2439)

D. VARIABLE TRANSFORMATIONS USING SUMT CODE

Experiment Number 28 HIMMELBLAU PROBLEM 16

f(x*)=.8660277 67 points CPU=11.69 sec.

$\underline{x}^*$=(-.4924,-.3562,.5076,-.8616,-.4924,-.8703,.5076,-.3475,.5141)

Experiment Number 29 HIMMELBLAU PROBLEM 16

Place lower bounds on all variables

$f(\underline{x}^*)=.8660196$ 59 points CPU=15.29 sec.

$\underline{x}^*=(.9753,.2208,.2965,.9550,.9753,.2208,.2964,.9550,0.0)$

Experiment Number 30 HIMMELBLAU PROBLEM 16

Transformation: $x_i=y_i^2$

$f(\underline{x}^*)= .8660178$ 70 points CPU=15.78 sec.

$\underline{x}^*= (.8660,.4999,.0016,.9998,.8660,.4985,.0003,1.0,0.0)$

Experiment Number 31 HIMMELBLAU PROBLEM 20

$f(\underline{x}^*)=.06520525$

Program was terminated after four minutes of CPU time without having reached optimum point.

Experiment Number 32 HIMMELBLAU PROBLEM 4

Transformation: $x_i=e^{y_i}$

$f(x^*)= -47.76488$ 57 points CPU=13.22 sec.

E. PRODUCT TRANSFORMATIONS USING GRG CODE-HIMMELBLAU PROBLEM 16

Experiment Number 33

Transformation: $y_i^2-y_j^2=x_ix_j$

$f(\underline{x}^*)= .43302779$ iterations=12 CPU=33.51 sec.

$\underline{x}^*= (.8660,.5000,.8660,.5000,0.0,0.0,.8660,1.5000,1.0,.....)$

Experiment Number 34

Transformation: $y_i^2 - y_j^2 = x_i x_j$

TOL parameter changed in DEGEN subroutine to 10^{-4}

f($\underline{x}$*)=.43302779 iterations=12 CPU=33.7 sec.

$\underline{x}$*=(.8660,.5000,.8660,.5000,0.0,0.0,.8660,1.5000,1.0,
.....)

F. SCALING EXPERIMENTS USING GRG-HIMMELBLAU PROBLEM 16

Experiment Number 35

Scaling Factors:

$x_1 \rightarrow x_3$: 100 $x_4 \rightarrow x_6$: 10 $x_7 \rightarrow x_9$: 1

f($\underline{x}$*)=.86603674 iterations=24 CPU=4.5 sec.

Experiment Number 36

Scaling Factors:

$x_1 \rightarrow x_5$: 10000 $x_6 \rightarrow x_9$: 1

No feasible point found

Experiment Number 37

Scaling Factors:

$x_1 \rightarrow x_3$: 1000 $x_4 \rightarrow x_6$: 100 $x_7 \rightarrow x_9$: 1

No feasible point found

Experiment Number 38

Scaling Factors:

$x_1 \rightarrow x_3$: 1 $x_4 \rightarrow x_6$: 10 $x_7 \rightarrow x_9$: 100

f(x*)= .86604509 iterations=19 CPU=3.24 sec.

Experiment Number 39

Scaling Factors

$x_1 \rightarrow x_3$: 10 $x_4 \rightarrow x_6$: 10 $x_7 \rightarrow x_9$: 10

$f(\underline{x}^*)$= .86600928 iterations=17 CPU=3.47 sec.

G. TRANSLATION EXPERIENT USING GRG-HIMMELBLAU PROBLEM 16

Experiment Number 40

Translation: $x_i = y_i + 0.5$

$y^0 = (0.5, 0.5, \ldots, 0.5)$

$f(\underline{x}^*)$= .86602981 iterations=19 CPU=3.88 sec.

H. ROTATION EXPERIMENTS USING GRG-HIMMELBLAU PROBLEM 16

Experiment Number 41

$x_1 = y_1 - y_2$

$x_2 = y_1 + y_2$

$f(\underline{x}^*)$=.86601039 iterations=18 CPU=3.37 sec.

Experiment Number 42

$x_3 = y_3 - y_4$

$x_4 = y_3 + y_4$

$f(\underline{x}^*)$=.86605021 iterations=18 CPU=3.17 sec.

Experiment Number 43

$x_5 = y_5 - y_6$

$x_4 = y_5 + y_6$

$f(\underline{x}^*)$=.86603622 iterations=19 CPU=3.39 sec.

Experiment Number 44

$x_7=y_7-y_8$

$x_8=y_7+y_8$

$f(\underline{x}^*)=.50000327$ iterations=16 CPU=3.24 sec.

Experiment Number 45

$x_8=y_8-y_9$

$x_9=y_8+y_9$

$f(\underline{x}^*)=.5001803$ iterations=21 CPU=3.49 sec.

Experiment Number 46

$x_1=y_1-y_9$

$x_9=y_1+y_9$

$f(\underline{x}^*)=.86604409$ iterations=18 CPU=3.20 sec.

Experiment Number 47

$x_1=y_1-y_2$ $x_3=y_3-y_4$ $x_5=y_5-y_6$ $x_7=y_7-y_9$

$x_2=y_1+y_2$ $x_4=y_3+y_4$ $x_6=y_5+y_6$ $x_9=y_7+y_9$

$f(\underline{x}^*)=.86600559$ iterations=18 CPU=3.13 sec.

Experiment Number 48

$\underline{x}=\underline{P}\ \underline{y}$

$$\underline{P} = \begin{pmatrix} 1 & -1 & -1 & 0 & 0 & 0 & 1 & 0 & -1 \\ 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 & -1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$f(\underline{x}^*)=.86602427$ iterations=20 CPU=3.25

I. SCALING EXPERIMENTS USING GRG-HIMMELBLAU PROBLEM 20

Experiment Number 49

Scaling Factors:

$x_1 \rightarrow x_8$: 10 $x_9 \rightarrow x_{16}$: 10 $x_{17} \rightarrow x_{24}$: 10

No feasible point found

Experiment Number 50

Scaling Factors:

$x_1 \rightarrow x_{12}$: 10 $x_{13} \rightarrow x_{24}$: 1

No feasible point found

Experiment Number 51

Scaling Factors:

$x_1 \rightarrow x_8$: 100 $x_9 \rightarrow x_{16}$: 10 $x_{17} \rightarrow x_{24}$: 1

No feasible point found

Experiment Number 52

Scaling Factors:

$x_1 \rightarrow x_8$: 2 $x_9 \rightarrow x_{16}$: 2 $x_{17} \rightarrow x_{24}$: 2

No feasible point found

Experiment Number 53

Scaling Factors:

$x_1 \rightarrow x_8$: 0.1 $x_9 \rightarrow x_{16}$: 0.1 $x_{17} \rightarrow x_{24}$: 0.1

$f(\underline{x}^*)$=.055658041 iterations=29 CPU=14.76 sec.

J. TRANSLATION EXPERIMENT USING GRG-HIMMELBLAU PROBLEM 20

Experiment Number 54

Translation: $x_i = y_i + .04$

$\underline{y}^0 = (0.0, \ldots\ldots\ldots\ldots, 0.0)$

$f(\underline{x}^*) = .055658041$ iterations=30 CPU=15.01 sec.

K. ROTATION EXPERIMENTS USING GRG-HIMMELBLAU PROBLEM 20
(All lower bounds converted to inequality constraints)

Experiment Number 55

$x_1 = y_1 - y_2$

$x_2 = y_1 + y_2$

$f(\underline{x}^*) = .055658019$ iterations=26 CPU=19.61 sec.

Experiment Number 56

$x_3 = y_3 - y_4$

$x_4 = y_3 + y_4$

$f(\underline{x}^*) = .055658019$ iterations=27 CPU=19.89 sec.

Experiment Number 57

$x_5 = y_5 - y_6$

$x_6 = y_5 + y_6$

$f(\underline{x}^*) = .076972898$ iterations=24 CPU=15.93 sec.

Experiment Number 58

$x_7=y_7-y_8$

$x_8=y_7+y_8$

$f(\underline{x}^*)=.055658019$ iterations=29 CPU=20.45 sec.

Experiment Number 59

$x_9=y_9-y_{10}$

$x_{10}=y_9+y_{10}$

$f(\underline{x}^*)=.055658019$ iterations=27 CPU=19.05 sec.

Experiment Number 60

$x_1=y_1-y_2$ $x_3=y_3-y_4$ $x_5=y_5-y_6$ $x_7=y_7-y_8$

$x_2=y_1+y_2$ $x_4=y_3+y_4$ $x_6=y_5+y_6$ $x_8=y_7+y_8$

No feasible point; constraints violated = 4, 9

Experiment Number 61

$x_9=y_9-y_{10}$ $x_{11}=y_{11}-y_{12}$ $x_{13}=y_{13}-y_{14}$ $x_{15}=y_{15}-y_{16}$

$x_{10}=y_9+y_{10}$ $x_{12}=y_{11}+y_{12}$ $x_{14}=y_{13}+y_{14}$ $x_{16}=y_{15}+y_{16}$

No feasible point; constraints violated: 8,9,11,12

Experiment Number 62

$x_{17}=y_{17}-y_{18}$ $x_{19}=y_{19}-y_{20}$ $x_{21}=y_{21}-y_{22}$ $x_{23}=y_{23}-y_{24}$

$x_{18}=y_{17}+y_{18}$ $x_{20}=y_{19}+y_{20}$ $x_{22}=y_{21}+y_{22}$ $x_{24}=y_{23}+y_{24}$

No feasible point; constraints violated: 4,9,11,12

Experiment Number 63

$x_1=y_1-y_2$ $x_3=y_3-y_4$ $x_7=y_7-y_8$ $x_9=y_9-y_{10}$

$x_2=y_1+y_2$ $x_4=y_3+y_4$ $x_8=y_7+y_8$ $x_{10}=y_9+y_{10}$

$f(\underline{x}^*)=.11085991$ iterations=25 CPU=17.37 sec.

Experiment Number 64

Pairwise Rotation of all 24 variables

No feasible point found; constraint violated: 4

APPENDIX C

Generalized Reduced Gradient Method

The generalized reduced gradient method is an algorithm that solves nonlinear programming problems of the form:

$$\begin{aligned} &\text{minimize} && f(\underline{x}) \\ &\text{subject to:} && h(x)=0,\ \underline{a} \leq \underline{x} \leq \underline{b} \end{aligned} \tag{40}$$

where $h(\underline{x})$ is of dimension m.

The reduced gradient method was originally proposed by Wolfe [36] for problems with linear constraints and was generalized to handle nonlinear constraints by Abadie and Carpentier [37]. The material in this appendix is based on Himmelblau [26].

Inequality constraints are adjusted to the formulation above by the introduction of non-negative slack variables. The slack variables are added to the set of n variables. If a nondegeneracy assumption holds, the GRG algorithm partitions the variables into two distinct sets. One set consists of m basic, dependent variables, $\underline{x}_I$. The other set comprises (n-m) nonbasic, independent variables, $\underline{x}_K$.

At each iteration, the reduced gradient method considers the problem only in terms of the independent variables. Since the dependent variables are determined implicitly by the independent variables, the objective function is a function of the (n-m) independent variables.

The basic idea can be illustrated by the following example:

$$\begin{aligned} &\text{minimize} && f(x_1,x_2) \\ &\text{subject to:} && h(x_1,x_2)=0 \end{aligned} \tag{41}$$

For differential displacements in x_1 and x_2

$$\begin{aligned} df(\underline{x}) &= \frac{\partial f(\underline{x})}{\partial x_1}\, dx_1 + \frac{\partial f(x)}{\partial x_2}\, dx_2 \\ dh(\underline{x}) &= \frac{\partial h(\underline{x})}{\partial x_1}\, dx_1 + \frac{\partial h(\underline{x})}{\partial x_2}\, dx_2 = 0 \end{aligned} \tag{42}$$

Solve $dh(\underline{x}) = 0$ for dx_2

$$dx_2 = -\frac{\partial h(\underline{x})/\partial x_1}{\partial h(\underline{x})/\partial x_2}\, dx_1 \tag{43}$$

and introduce dx_2 into the differential objective function

$$df(\underline{x}) = \left(\frac{\partial f(\underline{x})}{\partial x_1} - \frac{\partial f(\underline{x})}{\partial x_2}\,\frac{\partial h(\underline{x})/\partial \underline{x}_1}{\partial h(\underline{x})/\partial x_2}\right) dx_1 \tag{44}$$

to yield the reduced gradient:

$$\frac{df(\underline{x})}{dx_1} = \frac{\partial f(\underline{x})}{\partial x_1} - \frac{\partial f(\underline{x})}{\partial x_2}\left[\frac{\partial h(\underline{x})}{\partial x_2}\right]^{-1}\frac{\partial h(\underline{x})}{\partial x_1} \tag{45}$$

One necessary condition for $f(\underline{x})$ to be minimum is:

$$\frac{df(\underline{x})}{dx_1} = 0 \tag{46}$$

Thus the generalized reduced gradient can be expressed as:

$$\frac{\partial f(\underline{x})}{\partial \underline{x}_K} = \nabla_{x_K}{}^T f - \nabla_{x_I}{}^T f \left(\frac{\partial h}{\partial x_I}\right)^{-1}\left(\frac{\partial h}{\partial x}{}_K\right) \tag{47}$$

As an example of this method consider the following problem:

$$\text{minimize} \quad x_1^2+x_2^2+x_3^2+x_4^2-2x_1-3x_4$$

$$\begin{aligned} \text{subject to:} \quad & 2x_1+x_2+x_3+x_4=7 \\ & x_1+x_2+2x_3+x_4=6 \\ & x_i \geq 0, \qquad i=1,2,3,4 \end{aligned} \tag{48}$$

Select x_1, x_2 as the basic variables. Then

$$\nabla_{x_I}^T f = \begin{bmatrix} 2x_1-2 & , & 2x_2 \end{bmatrix}$$

$$\nabla_{x_K}^T f = \begin{bmatrix} 2x_3 & , & 2x_4-3 \end{bmatrix}$$

$$\frac{\partial h}{\partial x_I} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \quad , \left(\frac{\partial h}{\partial x_I}\right)^{-1} = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}$$

$$\frac{\partial h}{\partial x_K} = \begin{bmatrix} 1 & 4 \\ 2 & 1 \end{bmatrix}$$

The reduced gradient becomes

$$\frac{\partial f(\underline{x})}{\partial x_K} = \begin{pmatrix} 2x_3, & 2x_4-3 \end{pmatrix} - \begin{pmatrix} 2x_1-2, & 2x_2 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix} \begin{pmatrix} 1 & 4 \\ 2 & 1 \end{pmatrix} \tag{49}$$

Simplifying and setting the reduced gradient equal to zero yields:

$$\begin{aligned} 2x_1 -6x_2 +2x_3-2=0 \\ -6x_1+ 4x_2 +2x_4+3=0 \end{aligned} \tag{50}$$

Given the feasible point $\underline{x}=(2,2,1,0)$ the reduced gradient equals $(-8,-1)$. This indicates that at this point, x_3 and x_4 are increased together in the ratio of eight to one. As they increase, x_1 and x_2 increase in such a way as to keep the constraints satisfied.

The algorithm continues as long as the reduced gradient is not zero. For each new iteration the $\underline{x}$ vector is changed by the relation:

$$\underline{x}^{(k+1)} = \underline{x}^{(k)} + \underline{\alpha}^{(k)} \underline{d}^{(k)} \tag{51}$$

where $\underline{\alpha}^{(k)} \geq 0$ is the step length, and $\underline{d}^{(k)}$ is the direction vector for the next iteration. The elements of the direction vector are determined differently for the independent and the dependent variables.

The search directions for the independent variables are determined as follows:

$$d_j^{(k)} = \begin{cases} 0 & , \text{ if } x_j^{(k)} = b_j \text{ and } z_j^{(k)} > 0 \\ 0 & , \text{ if } x_j^{(k)} = a_j \text{ and } z_j^{(k)} < 0 \\ -z_j^{(k)} & , \text{ if } a_j < x_j^{(k)} < b_j \end{cases} \tag{52}$$

where z_j is the j^{th} element of the reduced gradient; a_j is the lower bound on x_j; and b_j is the upper bound on x_j.

The search directions for the dependent variables are determined by linearizing the constraints:

$$\underline{d}_{x_I}^{(k)} = - \left(\frac{\partial \underline{h}}{\partial \underline{x}_I} \right)^{-1} \left(\frac{\partial \underline{h}}{\partial \underline{x}_K} \right) \underline{d}_{x_K}^{(k)} \tag{53}$$

The step size parameter is determined by a unidimensional dichotomous search. If the step size and step direction combination cause some elements of $\underline{x}_I^{(k)}$ to be infeasible, (denoted by $\hat{\underline{x}}_I^{(k)}$) then the dependent variables are adjusted to obtain a feasible $\underline{x}_I^{(k+1)}$. If at the point $(\underline{x}_K^{(k+1)}, \hat{\underline{x}}_I^{(k+1)})$

the constraints are not satisfied, a first order Taylor series approximation is made, and the resulting expression is solved for $\underline{x}_I^{(k+1)}$ as follows:

$$h\left(\underline{x}_K^{(k+1)}, \underline{x}_I^{(k+1)}\right) \approx h\left(\underline{x}_K^{(k+1)}, \hat{\underline{x}}_I^{(k+1)}\right) + \frac{\partial h\left(\underline{x}_K^{(k+1)}, \hat{\underline{x}}_I^{(k+1)}\right)}{\partial \underline{x}_I}\left(\underline{x}_I^{(k+1)} - \hat{\underline{x}}_I^{(k+1)}\right) = 0$$

$$\underline{x}_I^{(k+1)} - \hat{\underline{x}}_I^{(k+1)} = -\left(\frac{\partial \underline{h}}{\partial \underline{x}_I}\right)^{-1} h\left(\underline{x}_K^{(k+1)}, \hat{\underline{x}}_I^{(k+1)}\right) \qquad (54)$$

This last expression is termed an iteration by Newton's method and continues until one of the following outcomes occurs:

(1) If the last point obtained is feasible and there has been an improvement in the objective function, the Newton method is terminated and the search is continued starting with equation (51).

(2) If the last point obtained is feasible, but the objective function has worsened, the step size is reduced by some fraction and the Newton method is repeated.

(3) If the interations by Newton's method do not converge in a fixed number of iterations, the step size is reduced by some fraction and the Newton method is repeated.

(4) If the last point obtained is infeasible, a change in basis is carried out.

As can be seen from equation (51), if the step size is reduced to zero during any search iteration or during execution of the Newton method, the algorithm will terminate.

LIST OF REFERENCES

1. Case Western Reserve University Technical Memorandum 353, Design and Testing of a Generalized Reduced Gradient Code for Nonlinear Optimization, by L. Lasdon, A. Waren, A. Jain, and M. Ratner, March 1975.

2. Cleveland State University Technical Memorandum CIS-75-02, GRG User's Guide, by L. Lasdon, A. Waren, M. Ratner, and A. Jain, November 1975.

3. Fiacco, A.V. and McCormick, G.P., Nonlinear Programming: Sequential Unconstrained Minimization Techniques. John Wiley, 1968.

4. Research Analysis Corporation Report RAC-P-63, A Guide to SUMT-Version 4: The Computer Program Implementing the Sequential Unconstrained Mini-minization Technique for Nonlinear Programming, by W.C. Mylander, R.L. Holmes, and G.P. McCormick, October 1971

5. Stanford University Systems Optimization Laboratory Technical Report SOL 74-15, A Mini-Manual for Use of the SUMT Computer Program and the Factorable Programming Language, by Garth P. McCormick, August 1974.

6. Cleveland State University Technical Memorandum Number CIS-75-01, GRG System Documentation, by L. Lasdon, A. Waren, A. Jain, and M. Ratner, November 1975.

7. U.S. National Defense Research Committee Operations Evaluation Group Report No. 56, Theoretical Basis for Methods of Search and Screening, by B.O Koopman, v.2B, p.45-46, 1946.

8. Lee, S.M., Goal Programming for Decision Analysis, Auerbach, 1972.

9. Lee, S.M. and Moore, L.J., "Optimizing University Admissions Planning", Decision Sciences, v. 5, p.405-414, July, 1974.

10. Kuhn, H.W. and Tucker, A.W., "Nonlinear Programming" in Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, J.Neyman, ed., p.481-492, University of California Press, 1951.

11. Dantzig, G.B., Linear Programming and Extension, p. 471-472, and p. 482-490, Princeton, 1963.

12. Hillier, F. and Lieberman, G.J., Introduction to Operations Research, p. 575-578, Holden-Day, 1970.

13. Hadley, G., Nonlinear and Dynamic Programming, p. 104-144, and p. 185-205, Addison-Wesley, 1964.

14. Institute For Defense Analyses Research Paper P-661, A Convex Programming Model for Resourch Allocation With Time-Dependent Objectives: SLBM Attack of Bomber Bases, by J. Bracken and J. McGill, July,1970.

15. Institute For Defense Analyses Research Paper P-971, Defense Applications of Mathematical Programs With Optimization Problems in the Constraints, by J. Bracken and J. McGill, August, 1973.

16. Miller, C.E., "The Simplex Method for Local Separable Programming", in Recent Advances in Mathematical Programming, R.L. Graves and P. Wolfe, eds., p. 89-100, McGraw-Hill, 1963.

17. Beale, E.M.L., Mathematical Programming in Practice, p. 124-135, John Wiley, 1968.

18. Beale, E.M.L., "Numerical Methods", in Nonlinear Programming, J. Abadie ed., p. 173-188, John Wiley, 1967.

19. Charnes, A., Cooper, W.W., Klingman, D., and Niehaus, R.J., "Explicit Solutions in Convex Goal Programming," Management Science, v. 22, p. 438-448, December 1975.

20. Carnegie-Mellon University Management Science Report No. 341, Explicit Solutions in Convex Goal Programming, by A. Charnes, W.W. Cooper, D. Klingman, and R.J. Niehaus, June 1973.

21. Naval Postgraduate School Report 55Bz 75101, Large Scale Network Algorithms - Theory and Implementation, by G.H. Bradley, G.G. Brown, and G.W. Graves, October, 1975.

22. Stanford University Operations Research House Technical Report 73-9, Fractional Programming: Transformations, Duality and Algorithmic Aspects, by S. Schaible, November, 1973.

23. Charnes, A. and Cooper, W.W., "Programming with Linear Fractional Functionals," Naval Research Logistics Quarterly, v. 9, p. 181-196, September 1962.

24. Martos, B. "Hyperbolic Programming," Translated by A. and V. Whinston, Naval Research Logistics Quarterly, V. 11, p. 135-155, June, 1964.

25. Box, M.J., "A Comparison of Several Current Optimization Methods, and the Use of Transformations in Constrained Problems," The Computer Journal, v.9, p. 67-77, 1966.

26. Himmelblau, D., Applied Nonlinear Programming, p.274-290 and p.393-426, McGraw-Hill, 1972.

27. Wagner, H.M., Principles of Operations Research, p.551-557, Prentice-Hall, 1969.

28. The George Washington University Report T-267, Converting General Nonlinear Programming Problems to Separable Nonlinear Programming Problems, by G.P. McCormick, 16 June 1972.

29. Freeman, G.F., and Grabois, N.R., Linear Algebra and Multivariable Calculus, p.487-495, McGraw-Hill, 1970.

30. Luenberger, D.G., Introduction to Linear and Nonlinear Programming, p.278-301, Addison-Wesley, 1973.

31. Lootsma, F.A., Numerical Methods for Non-linear Optimization, p.313-343, Academic Press, 1972.

32. Pierre, D.A., Optimization Theory With Applications, p.333-344, John Wiley, 1969.

33. Zangwill, W.I., "Nonlinear Programming via Penalty Functions," Management Science, v.13, p.344-358, January, 1967.

34. Oren, S.S. and Luenberger, D.G., The Self-Scaling Variable Metric Algorithm (SSVM), paper presented at Fifth Hawaiian International Conference on System Scienses, January, 1972.

35. Oren, S.S., Self-Scaling Variable Metric Algorithms for Unconstrained Minimization, Ph.D. Dissertation, Dept. of Engineering-Economic Systems, Stanford University, Stanford, California, 1972.

36. Wolfe, P., "Methods of Nonlinear Programming," in Nonlinear Programming, J. Abadie ed., p.97-131, John Wiley, 1967.

37. Abadie, J. and Carpentier, J., "Generalization of the Wolfe Reduced Gradient Method to the Case of Nonlinear Constraints," in Optimization, R. Fletcher ed., p.37-47, Academic Press, 1969.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 55 Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
4.	Assoc. Professor G.H. Bradley, Code 55Bz Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	3
5.	Assoc. Professor G.G. Brown, Code 55Zr Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
6.	Assoc. Professor J.K. Hartman, Code 55Hh Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
7.	Lt. John Joseph Timar, USNR 620 West Seip Avenue Nazareth, Pennsylvania 18064	1

Thesis
T49 Timar
c.1 Modelling, transformations, and scaling decisions in constrained optimization problems.
165565